Magic xpa を Git リポジトリに接続

OUTPERFORM THE FUTURE™

本マニュアルに記載の内容は、将来予告なしに変更することがあります。これらの情報について MSE（Magic Software Enterprises Ltd.）および MSJ（Magic Software Japan K.K.）は、いかなる責任も負いません。

本マニュアルの内容につきましては、万全を期して作成していますが、万一誤りや不正確な記述があったとしても、MSE および MSJ はいかなる責任、債務も負いません。

MSE および MSJ は、この製品の商業価値や特定の用途に対する適合性の保証を含め、この製品に関する明示的、あるいは黙示的な保証は一切していません。

本マニュアルに記載のソフトウェアは、製品の使用許諾契約書に記載の条件に同意をされたライセンス所有者に対してのみ供給されるものです。同ライセンスの許可する条件のもとでのみ、使用または複製することが許されます。

当該ライセンスが特に許可している場合を除いては、いかなる媒体へも複製することはできません。ライセンス所有者自身の個人使用目的で行う場合を除き、MSE または MSJ の書面による事前の許可なしでは、いかなる条件下でも、本マニュアルのいかなる部分も、電子的、機械的、撮影、録音、その他のいかなる手段によっても、コピー、検索システムへの記憶、電送を行うことはできません。

サードパーティ各社商標の引用は、MSE および MSJ の製品に対するコンパチビリティに関しての情報提供のみを目的としてなされるものです。

本マニュアルにおいて、説明のためにサンプルとして引用されている会社名、製品名、住所、人物は、特に断り書きのないかぎり、すべて架空のものであり、実在のものについて言及するものではありません。

Magic は Magic Software Japan K.K. の登録商標です。
Magic xpa は Magic Software Enterprises Ltd. のイスラエルその他の国での商標または登録商標です。
Magic xpa Enterprise Studio、Magic xpa Enterprise Client、Magic xpa Enterprise Server および Magic xpa RIA Server は Magic Software Japan K.K. の商標です。

一般に、会社名、製品名は各社の商標または登録商標です。
MSE および MSJ は、本製品の使用またはその使用によってもたらされる結果に関する保証や告知は一切していません。この製品のもたらす結果およびパフォーマンスに関する危険性は、すべてユーザが責任を負うものとします。

この製品を使用した結果、または使用不可能な結果生じた間接的、偶発的、副次的な損害（営利損失、業務中断、業務情報の損失などの損害も含む）に関し、事前に損害の可能性が勧告されていた場合であっても、MSE および MSJ、その管理者、役員、従業員、代理人は、いかなる場合にも一切責任を負いません。

2022 年 8 月 31 日

このホワイトペーパーは、Magic xpa アプリケーションを開発する際 Git リポジトリを接続して利用する方法を説明することを目的としています。

TortoiseGit は Magic xpa のバージョン管理プロバイダです。Git への接続は TortoiseGit の Git プラグインを介して行います。Git を使用することで、チームでの Magic xpa アプリケーションのインストール、開発、実行、運用などを行うことができるようになります。

注意 : Git 対応は、現在サポート対象外です。

Git の紹介

ここでは、Git のバージョン管理システムを紹介します。

Git リポジトリ

他の MS SCC プロバイダとは異なり、Git は集中管理されたバージョン管理システムではありません。これは分散バージョン管理システムです。これらの 2 つの違いは、分散バージョン管理システムでは、すべての開発者が完全なバージョン履歴のローカルコピーを持ちますが、集中バージョン管理システムでは、バージョン履歴はサーバ側のリポジトリに格納されます。Git のバージョン管理下のすべてのプロジェクトには、1 つのローカルリポジトリと 1 つのリモートリポジトリがあります。

Git は、ワークフローの複雑さが大幅に増大したことに対して、柔軟性とパフォーマンスを提供します。

プロジェクトの開発を行う複数の独立した開発者グループが存在し、すべての開発者が中央のサーバに迅速にアクセスすることが難しいような場合は、Git を使用する必要があります。

Git を使用する利点は以下のとおりです。

- 高いパフォーマンス
- サーバから切断された場合でも機能する完全なローカルソースコントロール
- 履歴を失うことなく、サーバ間での作業に迅速に移行することができます。

チェックインとチェックアウトの操作はローカルで行います。そのため、他のユーザとの作業を同期させるためには、ローカルリポジトリからリモートリポジトリへの " プル " や " プッシュ " などの操作を行ったり、その逆を行ったりすることができます。

ソースファイルの準備

マージツールは Magic xpa 3.1 以降で作成されたソースファイルをサポートします。Magic xpa V2.x で作成されたされたソースファイルは古い構造のままです。

プロジェクトが Magic xpa 3.1 で作成されている場合は、リモートリポジトリに追加する前にソースファイルを更新する必要があります。これは、［オプション］メニューの［プロジェクトのソースファイルを更新］を選択することで行うことができます。

Git リポジトリでプロジェクトを開く

Git リポジトリの下でプロジェクトを開くには、以下の 2 つの方法があります。

Ⅰ .Magic xpa プロジェクトを Magic の外部から GIT に追加する

Magic xpa の外部から Git にプロジェクトを追加することができます。必要な作業は以下の通りです。

予め、Git Hub 上でリポジトリ（例：MergeTest）を作成しておいてください。

1. 新しいソリューションフォルダを作成します。（例：MyProject）
2. フォルダにプロジェクトを作成 / コピーします。例：TestProject）
3. Git リポジトリの初期化を行います (git init)。
4. .gitattributes と .gitignore ファイルをプロジェクトフォルダに追加します。これは手動で行う必要があります。
   ※この 2 つのファイルは、リモートリポジトリの作成後にソリューションフォルダ上にプロジェクトを作成した場合、自動的に作成されます。
5. ローカルリポジトリに追加するフォルダをステージします。（git add <folder_name>）

6. フォルダをローカルリポジトリに追加します（git commit -m "<Comment>"）。
7. Master ブランチを作成します。(git branch -M master)。
8. git remote add origin <git hub/git lab url> コマンドを実行します（例：git remote add origin https://github.com/<User_Name>/MergeTest.git）。
9. すでに、作成済みのリモートリポジトリを指定する場合は、git remote set-url origin <git hub/git lab url> を実行します。
10. 変更内容をリモートリポジトリにプッシュします (git push -u origin master または git push -f origin ma)。

ノート

ソリューションフォルダ上にプロジェクトをを作成 / コピーする処理は、リモートリポジトリの作成後に行うこともできます。

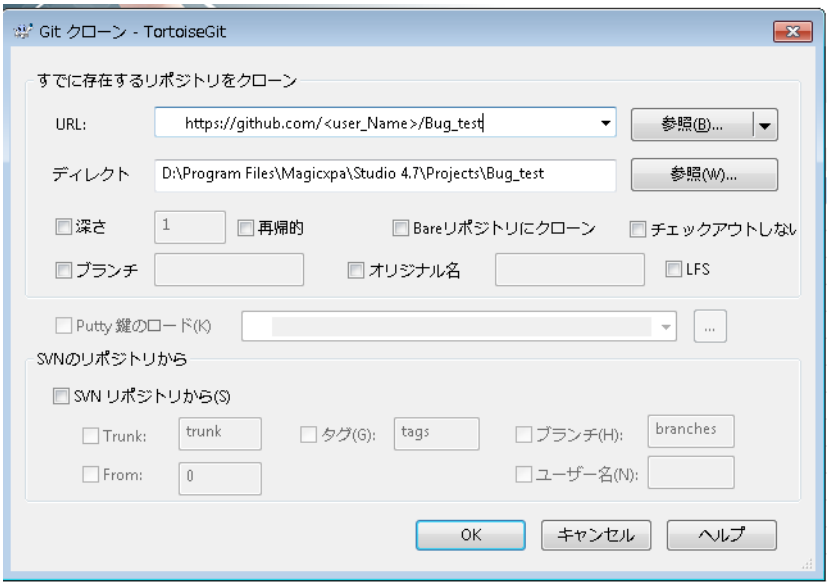
リモートリポジトリに接続する際に、Git Hub へのログインダイアログが表示される場合があります。Git Hub へのアカウントを入力してログインしてください。

Ⅱ .Magic xpa プロジェクトを Git リポジトリからクローンする

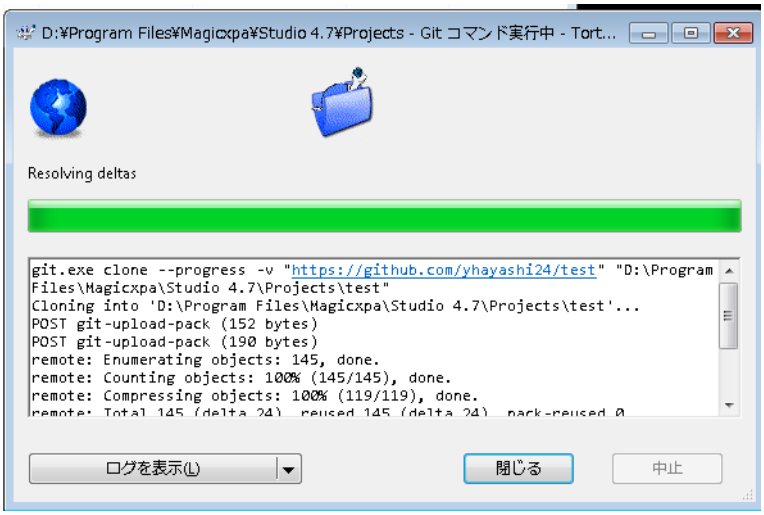
［メニュー］オプションの ［バージョン管理］> ［GIT リポジトリのクリーン］を使用してプロジェクトをクローンすることができます。

- ［URL］は、上記の #8 (git remote add origin ...) で指定した URL を指定します。

- ［ディレクトリ］は、クローンの保存先を指定します。



［OK］をクリックすると、コマンドの処理ダイアログが表示され、クローンが作成されます。



以下の手順は、Magic xpa プロジェクトのクローンを作成するための一般的な処理の流れを説明しています。

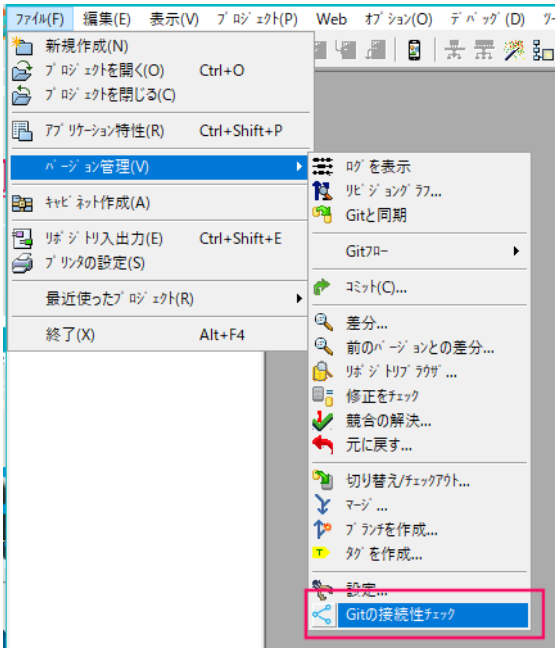
1. プロジェクトのクローン作成を実行します。
2. EDP ファイルを開きます。
3. ［Git の接続性］ダイアログが表示されます。ここで、状況を確認します。

ノート

プロジェクトの共有中にプッシュ / コミットが保留されていると、問題が発生する可能性があります。

Git との接続を確認する

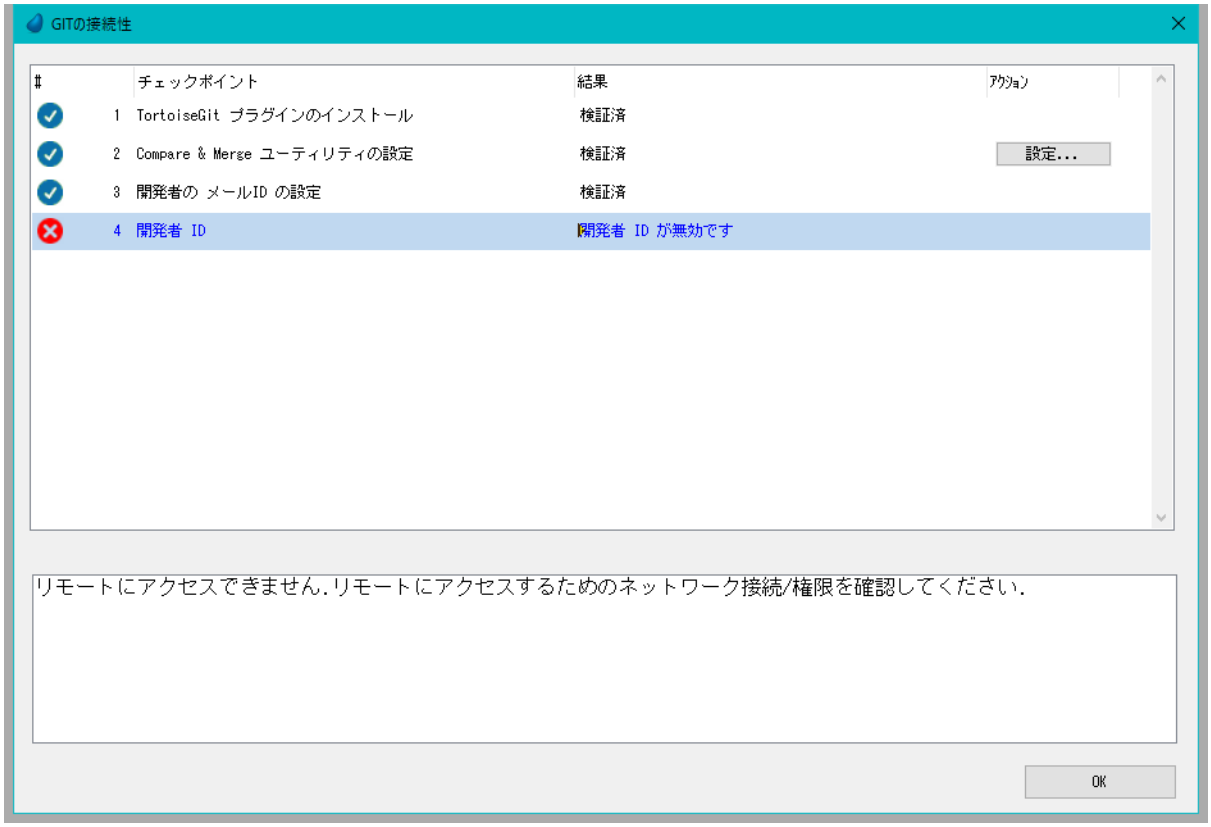
新しいプロジェクトを追加したり、既存のプロジェクトを Git リポジトリの配下で開いたりすると、Magic xpa からの Git 接続に特有の様々な必要なプラグインやユーティリティ、設定が存在するかどうかをチェックします。Git に接続しているかどうかは、［ファイル］メニューの［バージョン管理］>［Git 接続性チェック］から確認することができます。



［Git の接続性］チェックダイアログは、新しい Magic xpa プロジェクトが Git に初めて追加されたときにもデフォルトで表示されます。

ソリューション内で新しいプロジェクトを作成する場合は、開発者 ID は利用できません。このため、Git の接続性チェック時に Magic xpa は［Git 接続性］ダイアログの下部に以下のようなエラーを表示します。

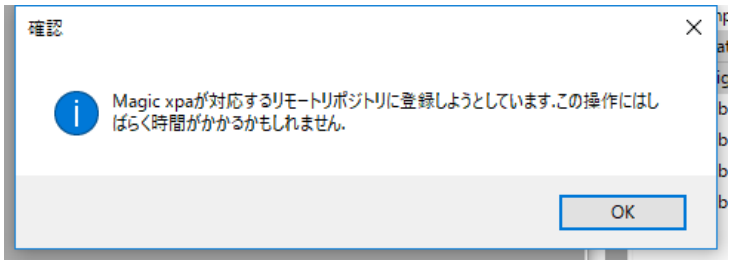
。



ノート

「Compare&Merge ユーティリティの設定」でエラーになる場合は、Merge.exe ファイルを選択するようにしてください。

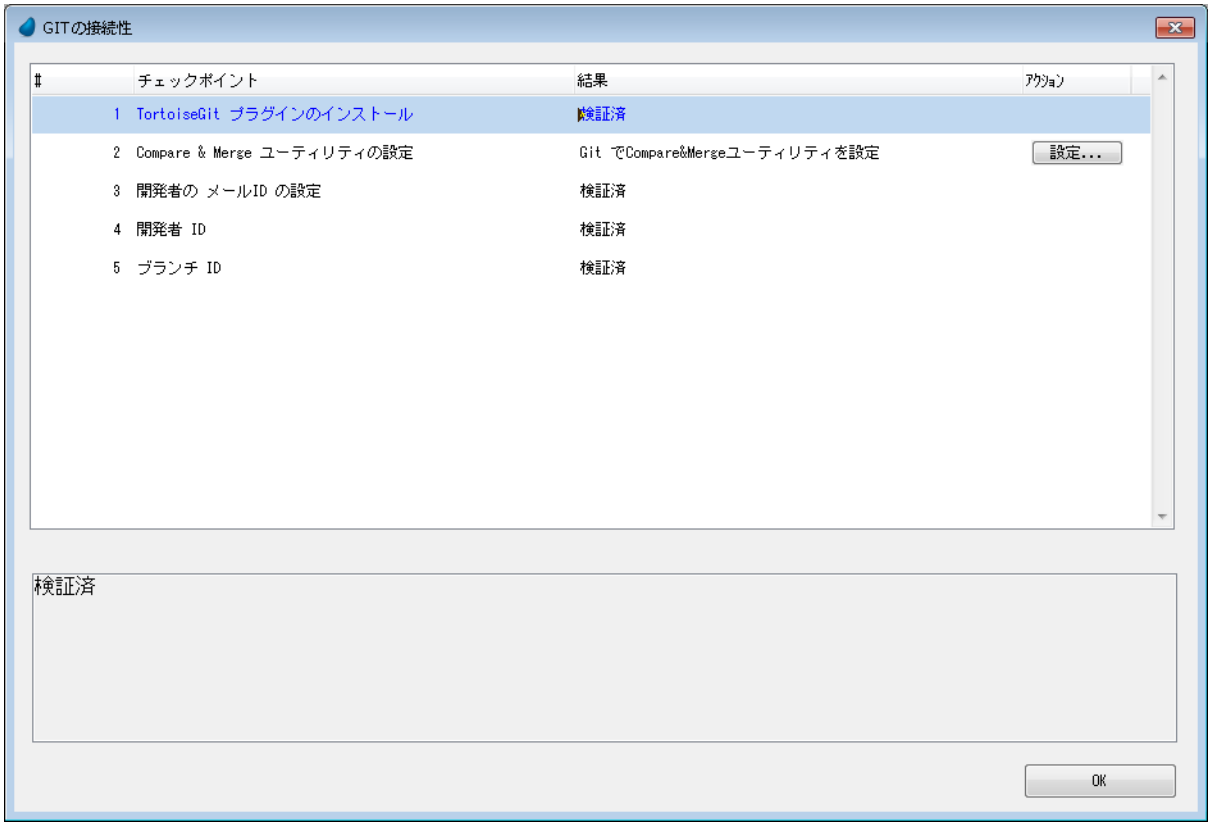
ソリューションからリモートリポジトリにプロジェクトを追加すると、登録処理が開始されます。Magic xpa は、リモートの Git リポジトリに新規ユーザーとして登録する際に、以下のようなメッセージダイアログが表示されます。



[OK] をクリックすると、開発者 ID が作成され、[Git の接続性] ダイアログが表示されます。エラーがなければ、リモートの Git リポジトリへの新規ユーザー登録が成功したことを意味します。

すでにこのリポジトリに登録している場合は、[Git の接続性] ダイアログは表示されずにプロジェクトが開きます。

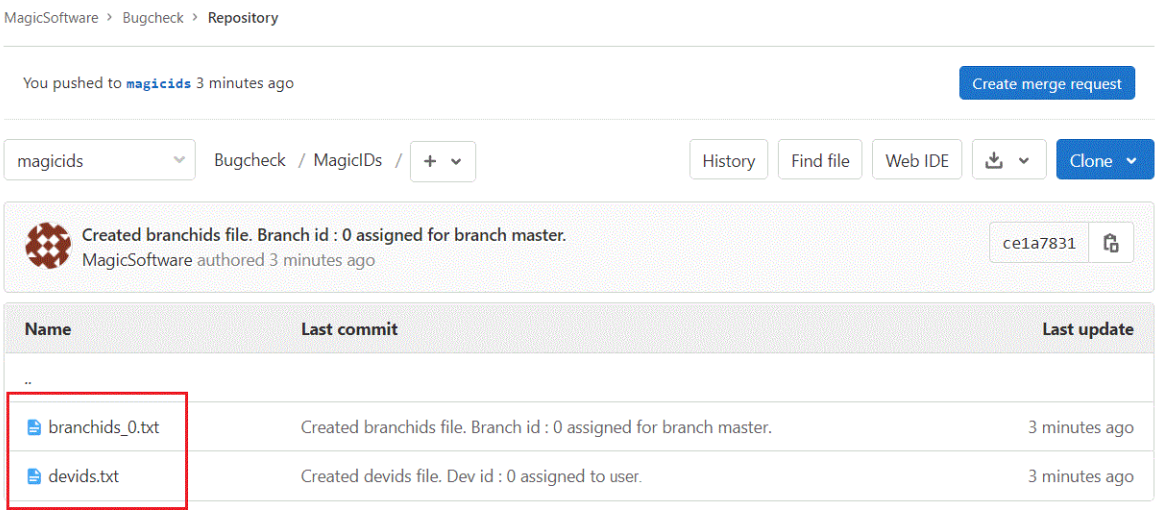
全てのオプションを確認して開発者 ID が作成されると、[Git の接続性] ダイアログは以下のように表示されます。



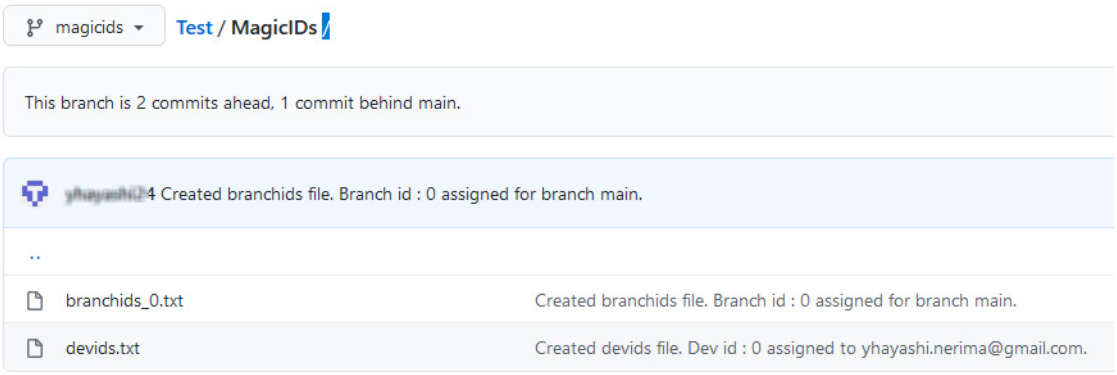
［ファイル］メニューの［バージョン管理］>［Git 接続性チェック］を選択した場合も同じ画面が表示されます。

Git Lab または Git Hub 上でリモートリポジトリを表示すると、以下に示すように、magicids という名前のブランチが表示されてます。その中に magicIDs というフォルダがあり、branchids_x.txt と devids.txt が含まれていることがわかります。

Git Lab の場合



Git Hub の場合



Git 接続性チェック時のチェックポイント結果

チェックポイント	結果
TortoiseGit プラグインのインストール	Magic xpa が TortoiseGit のインストールを見つけた場合は、" 確認済 " を表示します。
Compare&Merge ユーティリティの設定	Magic xpa が設定されたユーティリティを見つけた場合は、" 確認済 " を表示します。 そうでない場合は、Magic xpa はユーティリティを設定することができます。
開発者の メール ID の設定	Magic xpa がメールアドレスを検出した場合は、" 確認済 " を表示します。 メールアドレスが正しく入力されていない場合で設定ファイル (Git の設定) を変更すると、Magic xpa は新しい認証情報を把握して記録します。

チェックポイント	結果
開発者 ID	クローンプロジェクトを Magic xpa Studio で開いた時点でリモートリモートリポジトリに MagicIDs フォルダが作成されます。この中に devids.txt と branchids_x.txt ファイルが作成されると、" 確認済 " が表示されます。 後続のすべての開発者が、ソリューション内の同じプロジェクトで作業するために参加します。Magic xpa はそれぞれの ID を作成し、それぞれのプロジェクトを開くことができます。
ブランチ ID	Master ブランチ用に branchids_0.txt が作成された場合は、" 確認済 " を表示します。

ノート

Github は git の リモートコマンドを使用する前にリポジトリを作成する必要があります。

Magic Compare & Merge ツールの設定

Magic の Merge ツールの設定は、メニューの［バージョン管理］>［Git 接続性チェック］を使用して行います。Compare & Merge ユーティリティの設定 のオプションで Magic Merge ユーティリティを設定します。

リモート Git リポジトリの作成

リモートの Git リポジトリで作業するには、Git アカウントが必要です。

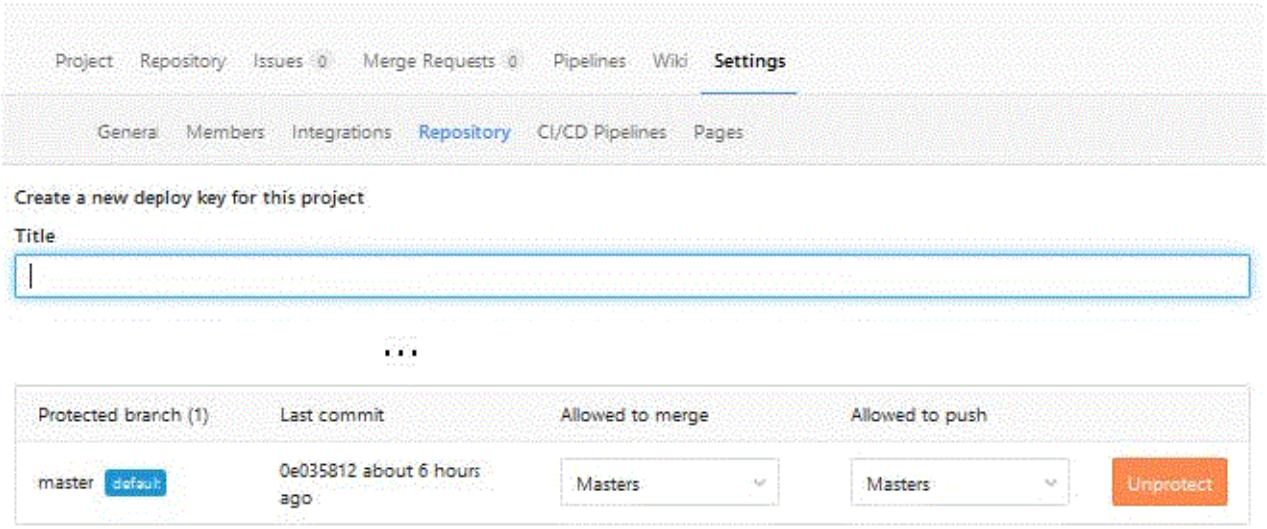
ノート

Git のホスティングサービスには GitHub や GitLab などがあります。GitHub や GitLab にインスタンスをホストして管理してもらうか、自分のサーバでホストすることができます。

このドキュメントでは、オンラインの Git サーバ（www.gitlab.com）を参照しています。このドキュメントに記載されている手順は、GitHub.com にも適用できます。

アカウントを定義したら、Git リポジトリ（GitHub の場合）または Git グループと Git プロジェクト（GitLab の場合）を作成する必要があります（例：myproj）。

GitLab を使用している間、Master ブランチを更新したい場合は、保護を解除する必要があります。プロジェクトの設定 (https://gitlab.com/Uid/myproj/settings/repository) にアクセスして、Master ブランチの Unprotect をクリックします。



.gitattributes と .gitignore ファイル

.gitattributes と .gitignore ファイルはプロジェクトフォルダに存在しなければなりません。

.gitattributes

.gitattributes ファイルは、マージ機能で利用されます。このファイルには、以下のエントリが含まれています。

```
*.xml MGXML merge=mgmerge
```

.gitignore

.gitignore ファイルは、Git が無視すべき意図的にトラックされていないファイルを指定します。.gitignore で指定したファイルは、コミットの対象とはなりません。.gitignore ファイルには、次のエントリが含まれています。

```
/Source/ProgramHeaders.xml

/Source/DataSourcesIndex.xml

*.log

*.lock

*.opt

*.sln

*.suo

*.xpaproj

*.xrf
```

リポジトリレベルの .gitignore ファイルには、以下のエントリが含まれているはずです。

```
MagicIDs/branchids*.*
```

これは、開発者がひとつのローカルリポジトリ上で開いたすべてのブランチのローカルリストを保持するためのもので、各ブランチにはこのリポジトリごとに一意の ID が与えられます。

このメカニズムは、開発者が同じリモートリポジトリを 2 つの異なるマシンでクローンしたり、同じマシン上の 2 つの異なるローカルリポジトリでクローンしたりする場合の解決策を提供しません。

ローカル処理の実行

コミットするコンテンツを選択したら、まずステージング（選択）する必要があります。

git add コマンドはコンテンツをバージョン管理の対象としてステージ化します。一度選択すると、それらはローカルリポジトリに移動します。

git commit コマンドは、変更した内容をローカルリポジトリに追加します。

コミットなどのローカル操作は、他のバージョン管理プロバイダを使用しているときに Magic xpa から行う場合と同じように、ローカルの Git データベース上で行われます。コミットされると、変更はローカルリポジトリにコミットされます。コミットされた変更は、ユーザがリモートリポジトリにプッシュするまで、他のユーザが利用することはできません。

Git でのブランチ

ここでは、ブランチ（分岐）の必要性と、ブランチを使用して様々な操作を行う方法を見ていきます。

ブランチの必要性

Git のような分散型のバージョン管理システムは、バージョン管理をどのように使用してコードを共有・管理するかという点で、個人に大きな柔軟性を与えています。チームは、この柔軟性と一貫した方法でコードを共同作業して共有する必要性との間にバランスを取る必要があります。

チームのメンバーは、他のメンバーと共有している Git ブランチを使用してコードの変更を公開したり、共有したり、レビューしたり、これらを繰り返したりすることができます。チームでブランチを採用することで、バージョン管理に費やす時間を減らし、コード開発に多くの時間を費やすことで、より良い共同作業ができるようにする必要があります。

たとえば、木を考えてみましょう。木には大きな幹にいくつかの枝があります。同様に、Git の Master ブランチは木の幹のようなもので、他の分岐はプロジェクトの小さな部分であり、Main ブランチ (Master を思い出してください) と干渉する必要はありません。これらのブランチには独自のコミットがあり、完了すると Master に追加されます。ほとんどの場合、Master ブランチに含まれるコードがデプロイされることになります。

ブランチの管理

Feature ブランチを使用して、ブランチの管理をシンプルにしましょう。Feature ブランチを作業の核とする考え方は、すべての機能開発は Master ブランチではなく、専用のブランチで行われるべきだということです。このようにカプセル化することで、複数の開発者がメインのコードベースに影響を与えることなく、特定の機能の開発を簡単に行うことができます。また、Master ブランチには壊れたコードが含まれていないことを意味します。

- すべての新機能、リリース、バグ修正には Feature ブランチを使用してください。
- プルリクエストを使用して、フィーチャー分岐を Master ブランチや Release ブランチにマージします。
- 高品質で最新の Master ブランチや Release ブランチを維持します。

これらのブランチにはそれぞれ特定の目的があり、どのブランチを元のブランチとし、どのブランチをマージ対象としなければならないかという厳格なルールに縛られています。メインのブランチやリリースブランチとは異なり、これらのブランチの寿命は常に限られています。

推奨事項

リモートディレクトリに直接ブランチを作成 / 定義することをお勧めします。そうすることで、各ブランチごとに作業フォルダを分けておくことができますし、ブランチの管理も簡単になります。

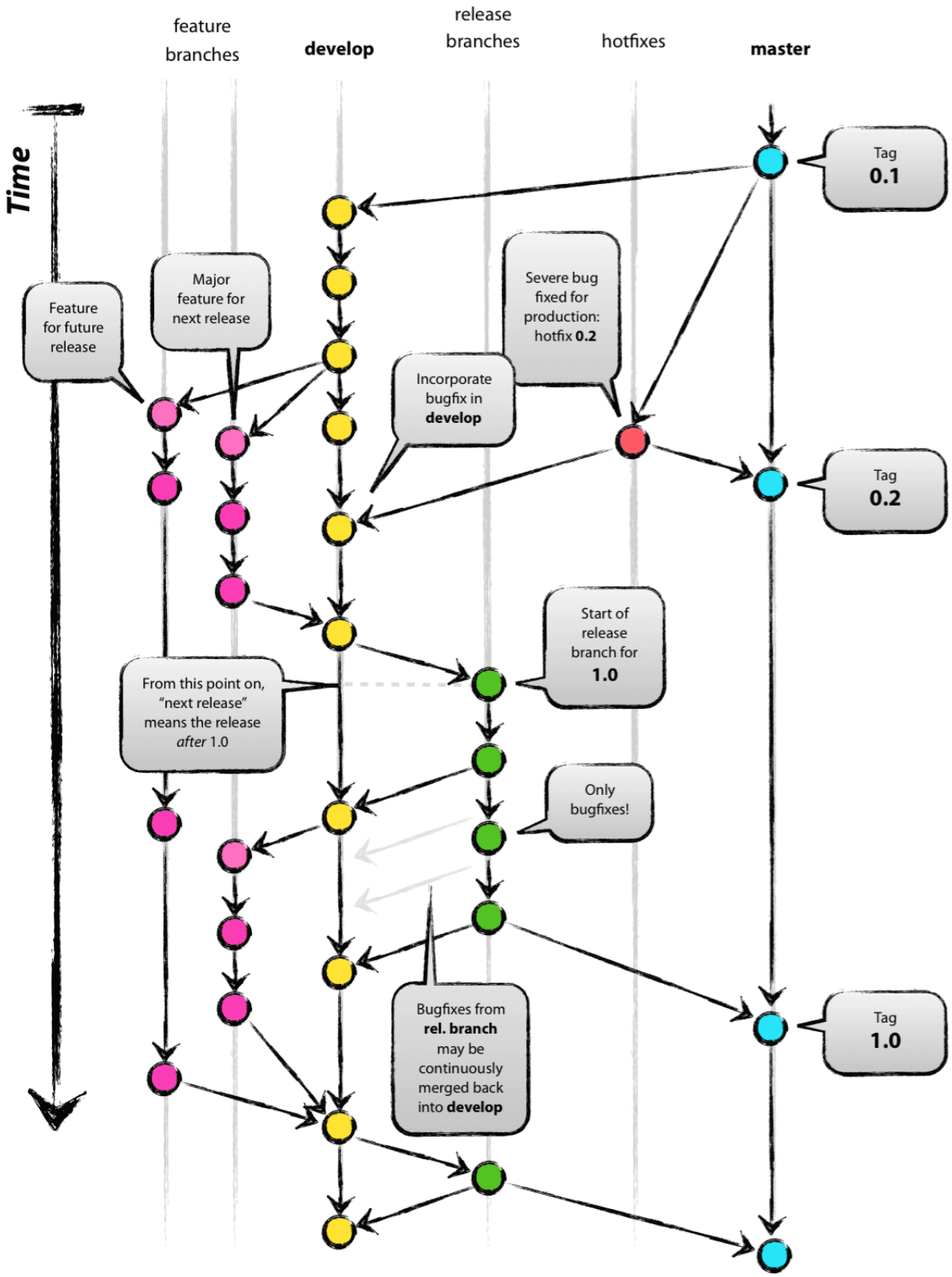
機能やバグフィックスごとにブランチを作成し、それをメインブランチにマージします。これはわずかなオーバヘッドになりますが、コーディングプロセスをクリーンに保つことができます。

以下の URL にあるガイドラインに従ってください。

- https://docs.microsoft.com/en-us/vsts/git/concepts/git-branching-guidance

- https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

ブランチのイラストはこのようになっています。



著者 : Vincent Driessen

オリジナルブログ :https://nvie.com/posts/a-successful-git-branching-model/

日本語訳 :https://qiita.com/homhom44/items/9f13c646fa2619ae63d0

各ブランチは以下のような役割を持っています。

- master …… プロダクトとしてリリースするためのブランチです。
- develop …… 開発用ブランチ。コードが安定し、リリース準備ができたら master へマージします。リリース前はこのブランチが最新バージョンとなります。
- feature …… 機能の追加用ブランチ。develop から分岐し、develop にマージします。
- hotfixes …… リリース後のクリティカルなバグフィックスなど、現在のプロダクトのバージョンに対する変更用ブランチ。master から分岐し、master にマージし、次に develop にマージします。
- release …… プロダクトリリースの準備用ブランチ。機能の追加やマイナーなバグフィックスとは独立させることで、リリース時に含めるコードを綺麗な状態に保つ（機能追加中で未使用のコードなどを含まないようにする）ことができます。

ユニークなバージョン管理 ID の生成

以前は、VersionControlID を決め Magic.ini で指定する必要がありました。VersionControlID と VCProjectID を覚えていないと、正しいプロジェクトで作業することができませんでした。このような場合、複数の開発者が作業をしていると、ISN の一意性が保証されません。Magic xpa では、ISN に自動的に一意性を持たせるために、ユニークな ISN を生成する機構を用意しています。

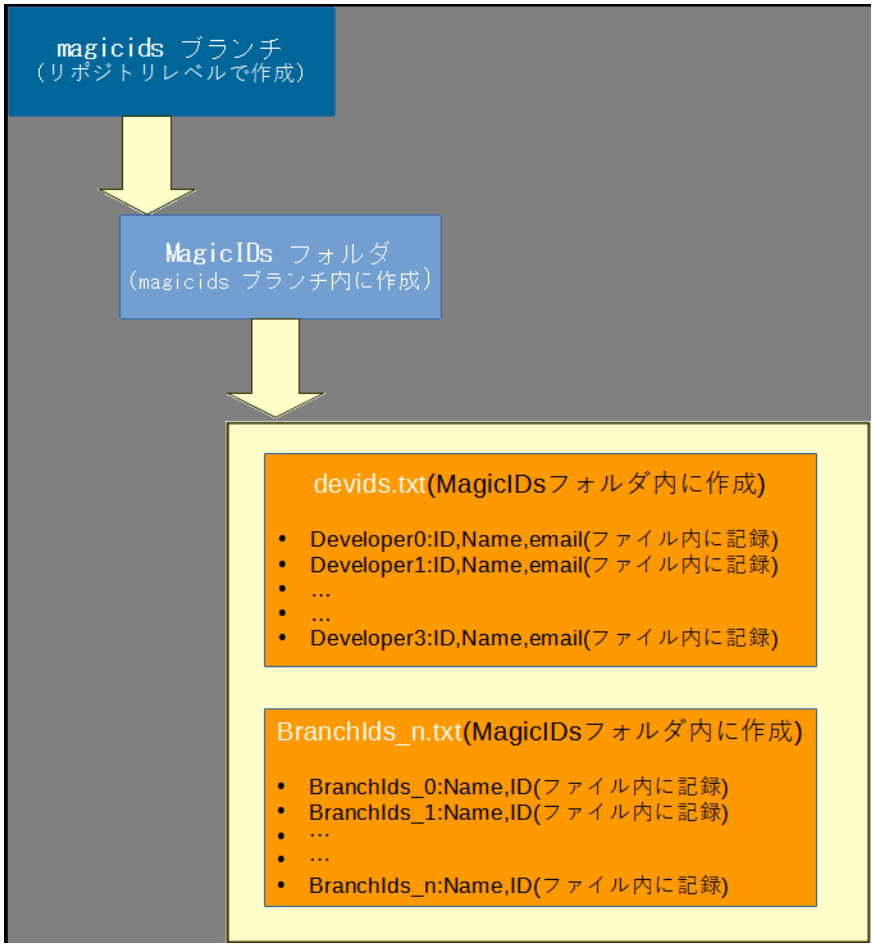
Magic xpa は、各開発者にユニークな ISN を割り当てるために、developer ID を作成します。この開発者 ID は、magicids という一意性ブランチの 'MagicIDs' というフォルダの中に作られる devids.txt というファイルに記録されます。

初めて Git に接続すると、Magic xpa は devids.txt ファイルに次の情報を追加します。

- 開発者の ID
- 開発者の名前
- 開発者の e メール

Magic xpa Studio 経由で Git プロジェクトを開くと、Magic xpa はユニークなファイルを読み込んで、開発者の ID とブランチ ID を設定して、そのプロジェクトの特定の開発者の ISN を作成します。

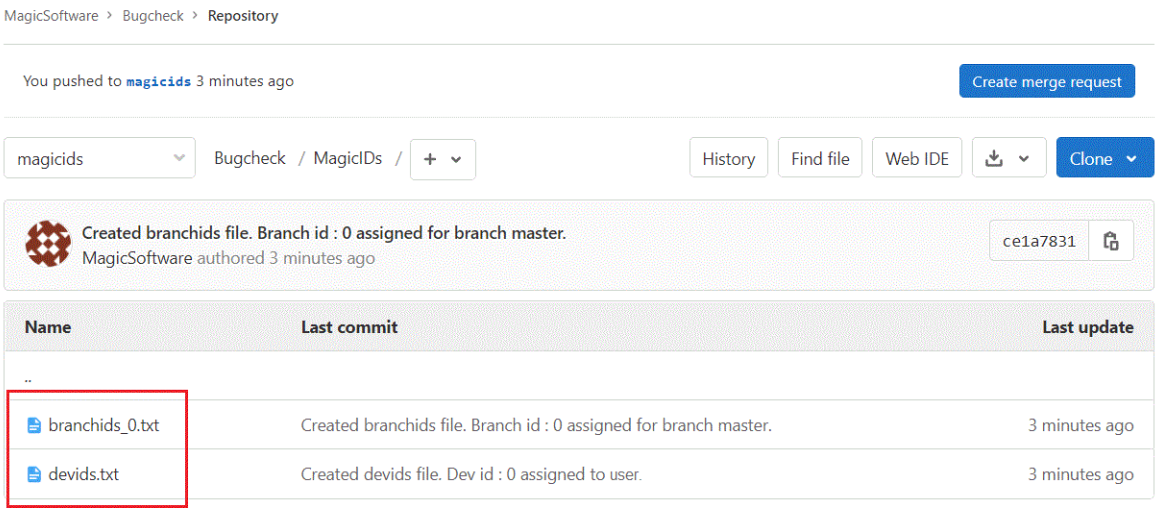
以下の図を参照してください。



devids.txt と branchIds_x.txt には、それぞれ一意の開発者 ID とブランチ ID が格納されています。

devids.txt ファイルはプロジェクトごとに作成されます。

Magic xpa は開発者ごとに branchids_x.txt を作成します。開発者と対応する作業ブランチに応じて、プロジェクトは Git で開かれます。ブランチが作成され、新しいブランチに切り替わるたびに、branchids_x.txt ファイル内のブランチ ID が更新されます。

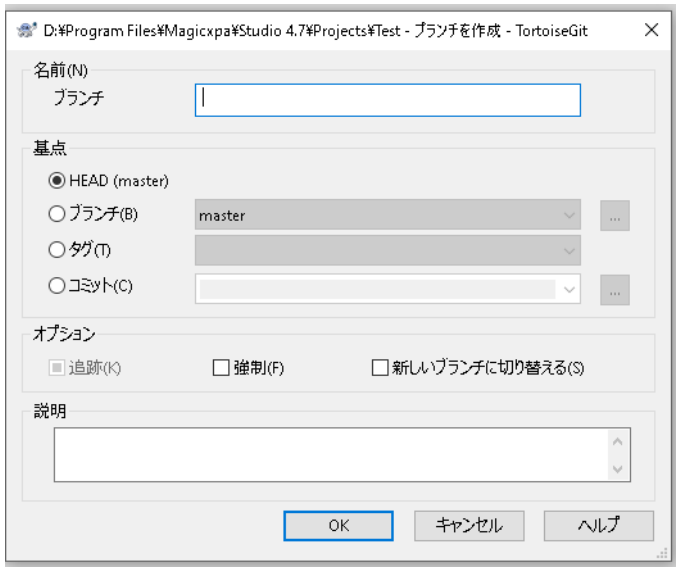


開発者が Magic xpa Studio 経由でプロジェクトを開くと、Magic.ini で定義されたローカルブランチのユニークなファイルにアクセスし、開発者の ID を設定します。

ユーザが変更されたりブランチが変更されたりすると、Magic xpa は自動的に branchids_x.txt と devids.txt ファイルにアクセスします。

Magic xpa から新しいブランチを追加する

メニューの ［バージョン管理］> ［ブランチを作成］を使用してブランチを追加することができます。



Magic xpa はユニークなブランチ ID を作成します。つまり、Magic xpa はリポジトリレベルで branchids_x.txt (x は開発者 ID) という名前のブランチ ID ファイル (ブランチでユニークなファイル) を MagicIDs という名前のフォルダ内に作成します。このファイルには、アクティブなブランチ名と番号が格納されています。

各ユーザごとに、ブランチ ID とブランチ名を含む特定の branchids_x.txt ファイルが作成されます。

ブランチが作成されるたびに、branchids_x.txt ファイルは更新されます。このファイルは、ブランチ作成時に存在しなかった場合に作成され、GIT リポジトリ内の一意性ブランチにプッシュされます。

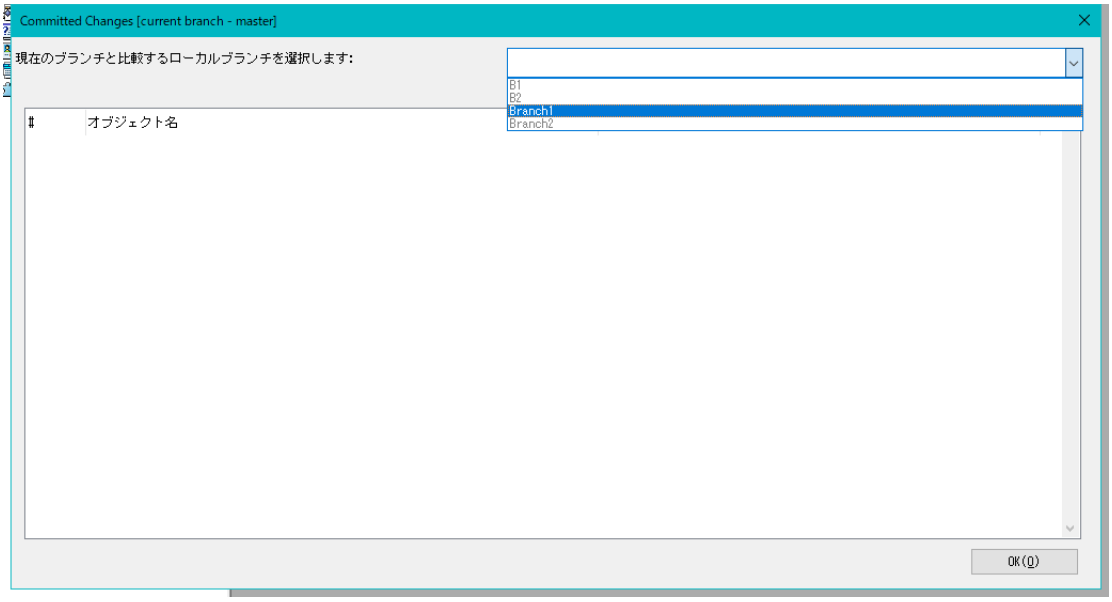
ノート

各ユーザは最大 64 個のブランチ ID を作成することができます。

ブランチを比較する

このオプションは、2 つの Git ブランチ（現在のブランチと選択したローカルブランチ）を比較することができます。

［ブランチの比較］メニューをクリックすると、ダイアログボックスが表示されます。

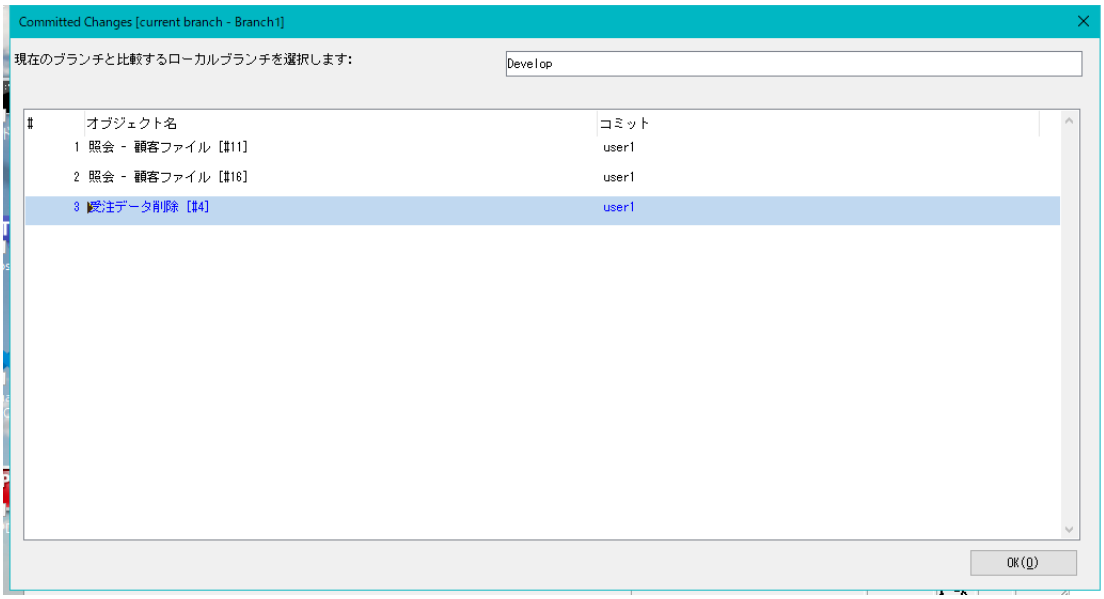


表示されているローカルブランチのリストから、ブランチを選択することができます。

2 つのブランチの比較結果は、［オブジェクト名］カラムに表示されます。ここには、開いているプロジェクトに属しているオブジェクトのみ表示されます。

オブジェクト上で右クリックすると、コンテキストメニューが表示されます。［バージョン管理 > 差分 ］のオプションをクリックすると、現在のブランチとローカルブランチの差分を確認することができます。

［コミット］カラムには、現在のブランチを最後にチェックインした人のユーザ ID（TortoiseGit のユーザ情報に設定した名前）が表示されます。



ノート

共通点のない無関係な 2 つのブランチを比較対象として選択した場合、その比較結果は期待通りのものにはなりません。

Magic Compare & Merge を使用したマージ処理

ブランチの作成方法がわかったところで、Git プラグインを使用した Magic xpa でのマージの仕組みを理解しておきましょう。

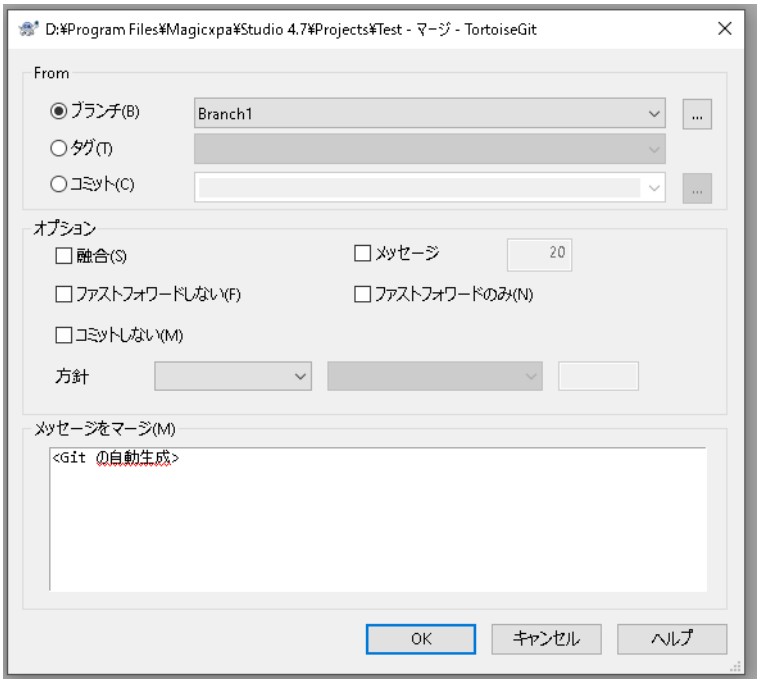
マージは、別のブランチで行われた変更を現在の作業ブランチに反映させるために使用されます。Magic xpa の柔軟で便利な機能のひとつに Magic Compare & Merge ツール があります。

ノート

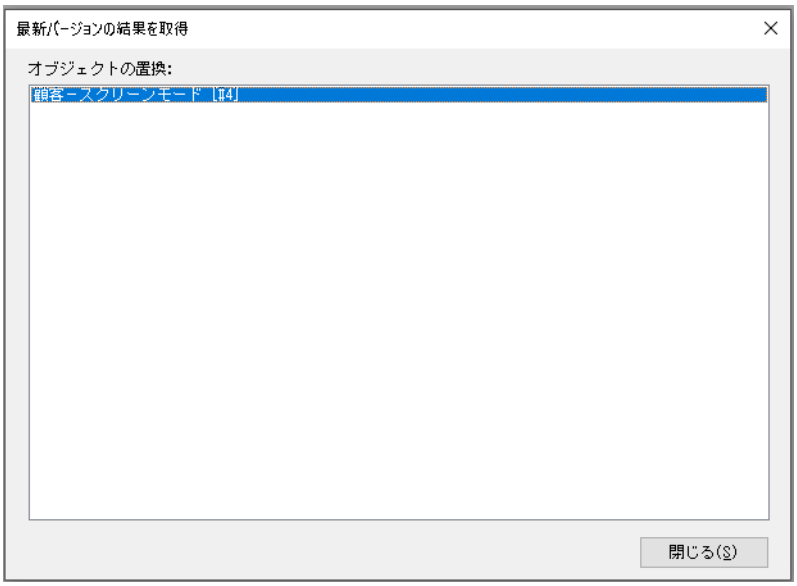
この時点で、Magic Merge ツールはすでに設定されている必要があります。マージツールを設定するには、Magic Compare & Merge ツール の設定を参照してください。

マージ操作

Branch1 ブランチ の変更内容を Master ブランチにマージする場合は、Master ブランチに切り替え、メニューの ［バージョン管理］> ［マージ］を選択するとマージ処理が実行されます。



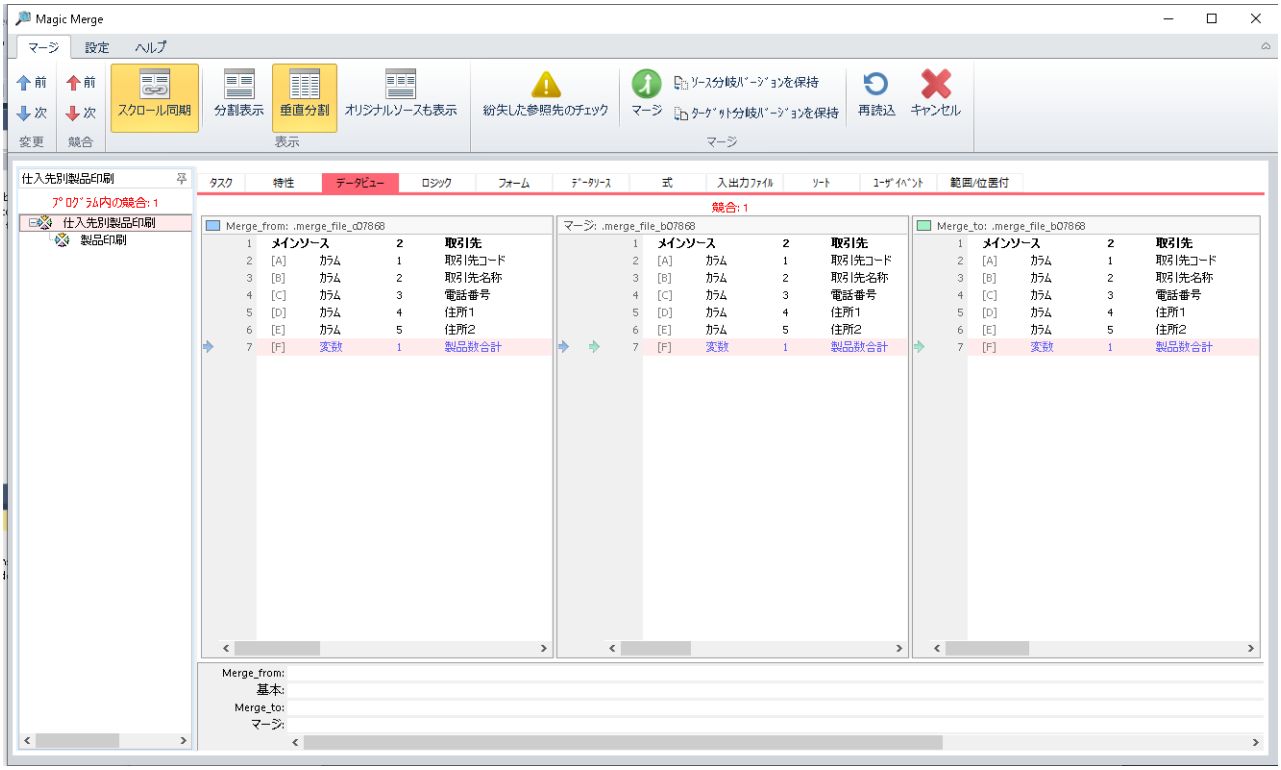
競合が発生しない場合は、即マージ処理が実行されます。



マージによって競合が確認された場合は、Magic Merge が起動されます。

［競合の修正］ボタンをクリックすると、Magic Merge のメイン画面が表示されます。

Magic Merge ツール は、プロジェクトのさまざまなコンポーネントに関連する変更内容を強調表示します。ソースおよび / またはターゲットごとに変更を確認し、反映したい項目の矢印をクリックして選択します。



競合を解消し、［マージ］アイコンをクリックしてマージ処理を実行します。マージが実行されコミットされます。

推奨事項

- 機能とバグ修正のために個別のブランチを作成し、それを Main にマージすることをお勧めします。この方法は、チームのバージョン管理のワークフローの矛盾を回避し、一貫性があり、従うのが簡単です。
- 機能やバグフィックスのような複数の変更に対して単一のプロジェクトで作業するのは避けましょう。これを行うことで、特定の機能やバグ修正のための単一ファイルのコミットではなく、ブランチコミットを行うことができます。

TortoiseGit プラグインでの作業

Magic xpa Studio 内で Git プロジェクトを管理するために、より包括的なサポートを提供するプラグインが必要とされています。

TortoiseGit は、一般的なプログラムと特定のプログラムをマージすることができ、より効率的にブランチから選択することができます。TortoiseGit は、Magic が Git リポジトリに接続するための最良の Git コネクタの候補になります。

TortoiseGit の前提条件

TortoiseGit をインストールするには、以下のものをあらかじめ用意しておく必要があります。

- Windows 8 以上
- Magic xpa Studio
- インストールには管理者権限が必要です。

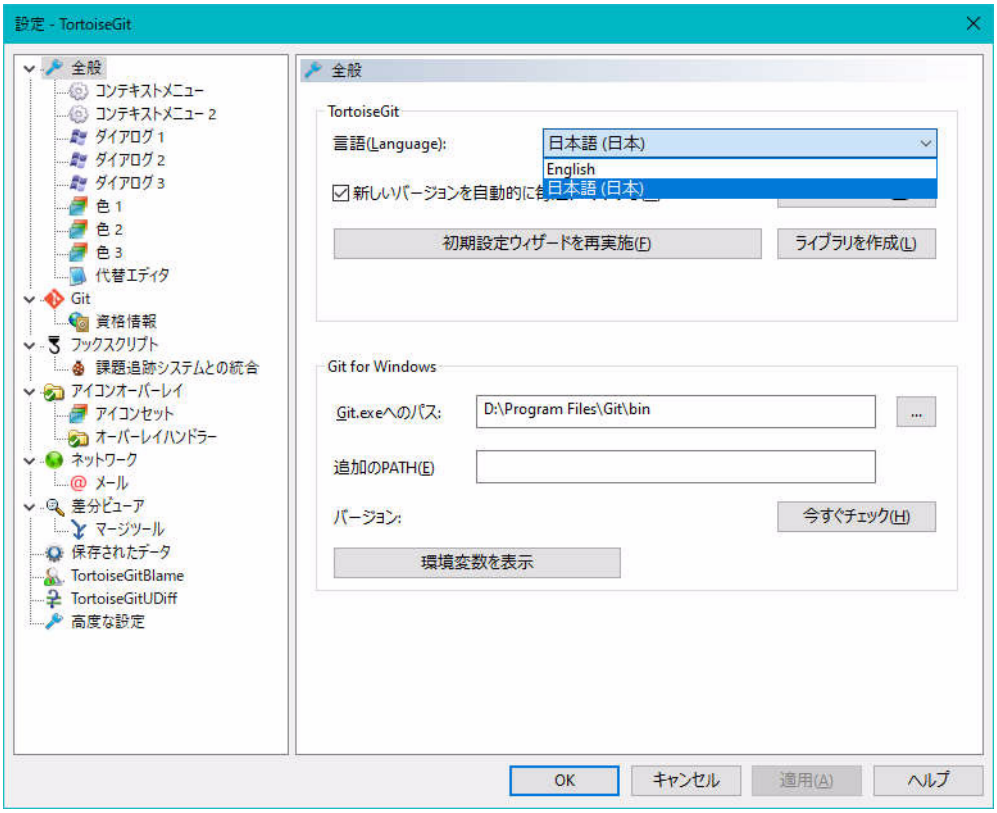
TortoiseGit のインストールと設定

TortoiseGit は、公式 Web サイト https://tortoisegit.org/download/ から、インストールファイルをダウンロードしてインストールできます。現在利用可能な安定版は 2.8.0 です。

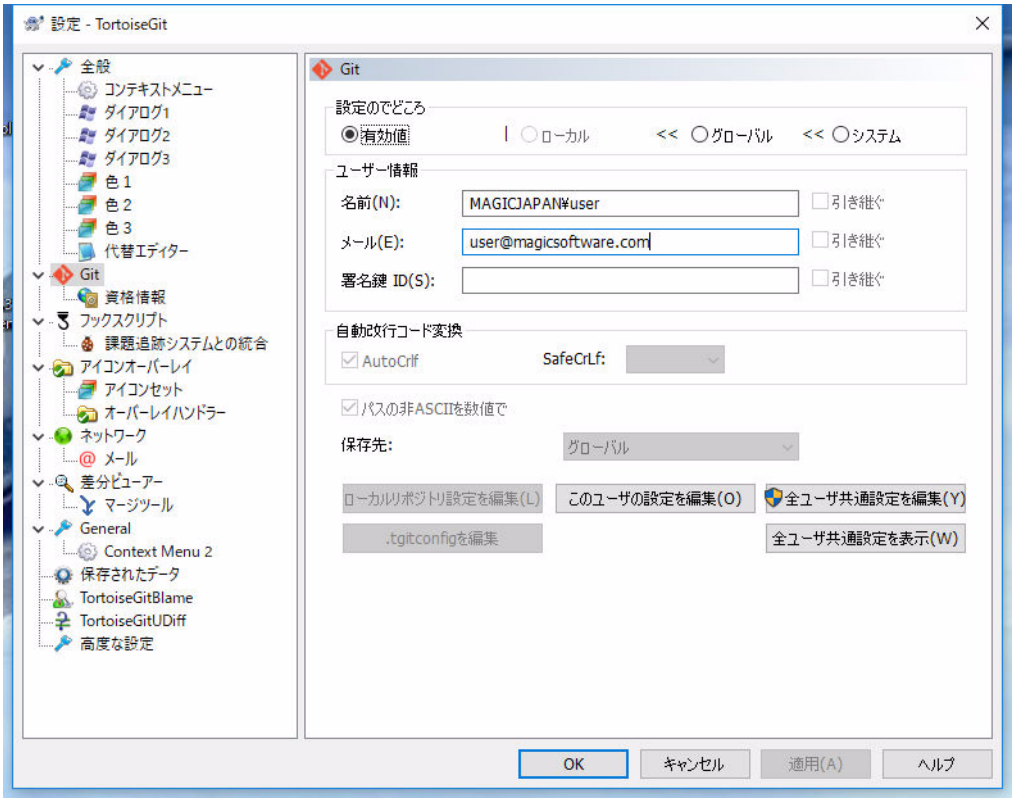
日本語表示を行う場合は、LanguagePack をインストールする必要があります。

Magic xpa Studio の［ファイル］メニューから［バージョン管理］>［Git の設定］を選択します。TortoiseGit の設定メニューが開くので、必要に応じて環境設定を行うことができます。

ツリーから［Genaral］を選択し、［Language］をクリックして、「日本語」を選択します。（LanguagePack をインストールすると有効になります）



ツリーから［Git］を選択し、［ユーザー情報］で［このユーザの設定を編集］をクリックして、Git Lab/Git Hub のアカウントを登録します。



TortoiseGitConfiguration の詳細については、次のサイトにアクセスしてください。https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html.

ノート

日本語版の "Language Pack" をインストールすることで、表示を日本語化することができます。

Magic xpa Studio での TortoiseGit ファイルベースのコマンド

Git には、Add や Delete などのファイルベースのコマンドはほとんどありません。

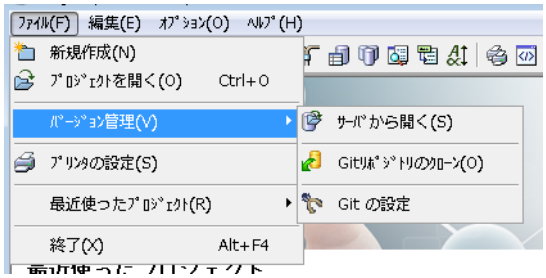
たとえば、新しいファイル / フォルダを作成し、ローカルリポジトリに追加することを決定できます。Git のバージョン管理のメニューの［追加］コマンドを使用すると、選択したファイル / フォルダーがローカル Git リポジトリに追加されます。後でコミットやプッシュの処理を実行することができます。

Magic xpa では、プロジェクト構造の性質により、これらのファイルベースの操作は、プログラム / タスクの作成時に内部的に処理されます。Magic xpa は、ローカルリポジトリへのファイルの追加と削除を内部的に処理します。さらに、Magic xp は、プログラムリポジトリの xml ファイルをを維持する手順を実行します。

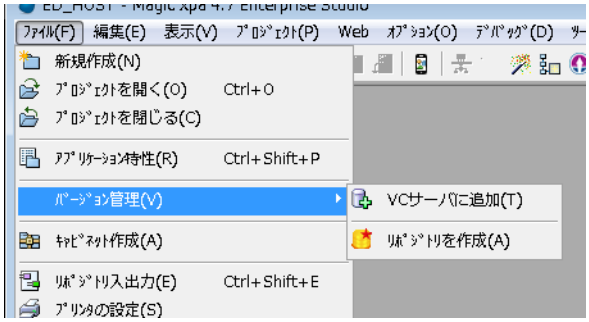
TortoiseGit メニュー

Git プラグインを選択した場合、Magic xpa Studio は、再起動後に関連するバージョン管理メニューを表示します。このメニューは、［ファイル］>［バージョン管理］を選択すると表示されます。

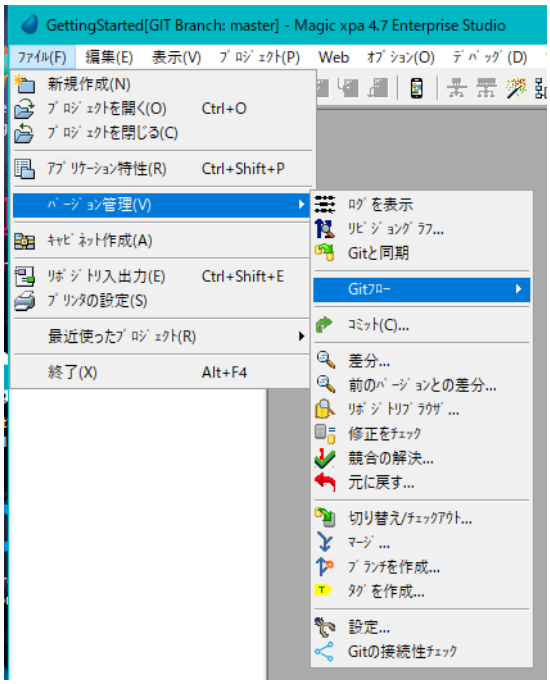
Magic xpa でプロジェクトを開いていない場合、［バージョン管理］では以下のメニューが表示されます。



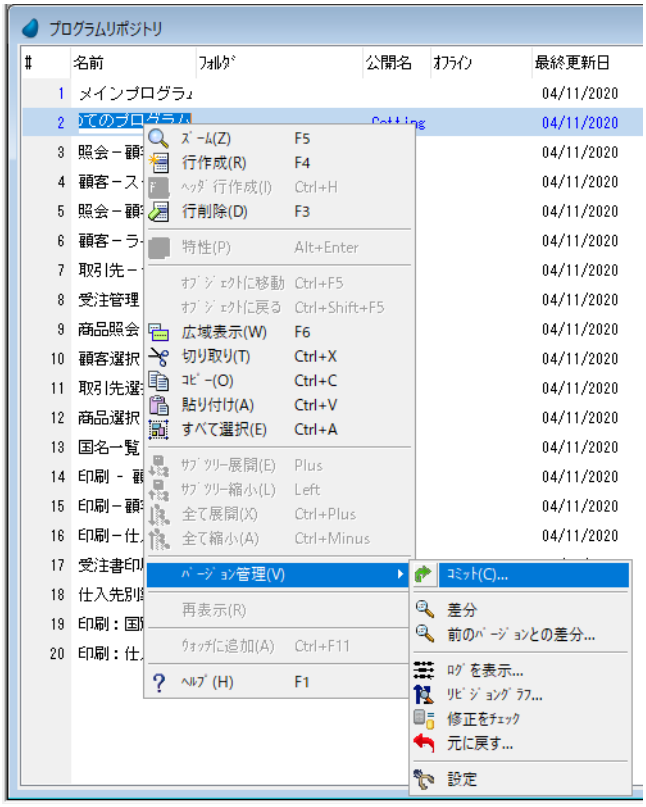
Magic xpa プロジェクトが TortoiseGit に関連付けられていなかった場合、以下のメニューが表示されます。



Magic xpa プロジェクトがすでに TortoiseGit に関連付けられている場合、以下のメニュー項目が表示されます。



以下のように、Magic xpa タスク上で右クリックすることで TortoiseGit 用のコンテキストメニューを開くことができます。



このメニューを使用して、リポジトリへの変更をコミットしたり、改訂ログ、改訂グラフを表示したり、行った変更を元に戻したりすることができます。

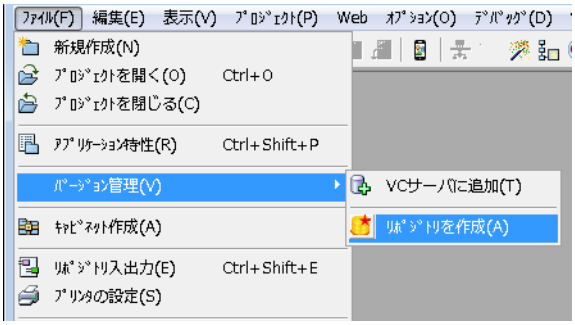
TortoiseGit でローカル処理を実行する

ローカル Git データベースで、コミット、ブランチ、更新などのローカル操作を実行することができます。

Git リポジトリの作成

プロジェクトがバージョン管理に関連付けられていない場合、リポジトリを作成する必要があります。以下のようにしてリポジトリを作成することができます。

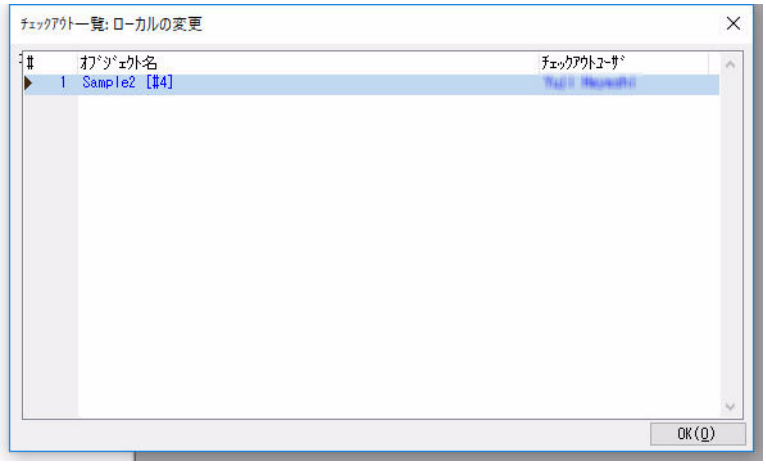
［ファイル］メニュー > ［バージョン管理］> ［リポジトリの作成］



修正のチェック

作業ツリーが修正されたかどうかを確認したい場合は、以下のようにします。

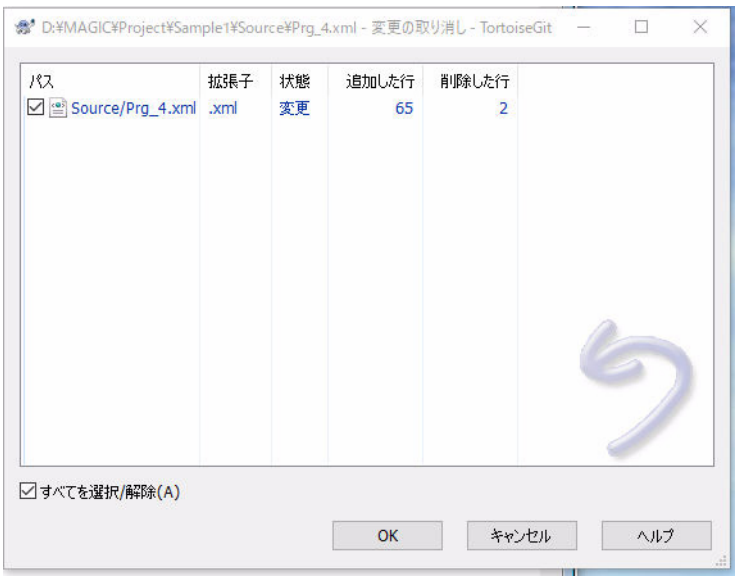
［バージョン管理］>［修正のチェック］を選択して［ローカルの変更］ダイアログを開きます。



元に戻す

このオプションにより、最後のコミット以降に行われたすべての変更を取り消すことができます。以下のようにします。

1. 変更したファイルを選択します。
2. 右クリックして、コンテキストメニューを開きます。
3. ［バージョン管理］>［元に戻す］を選択します。これにより、変更したファイルを含むダイアログボックスが開き、元に戻すことができます。
4. 元に戻す必要があるものを選択します。
5. ［OK］をクリックします。

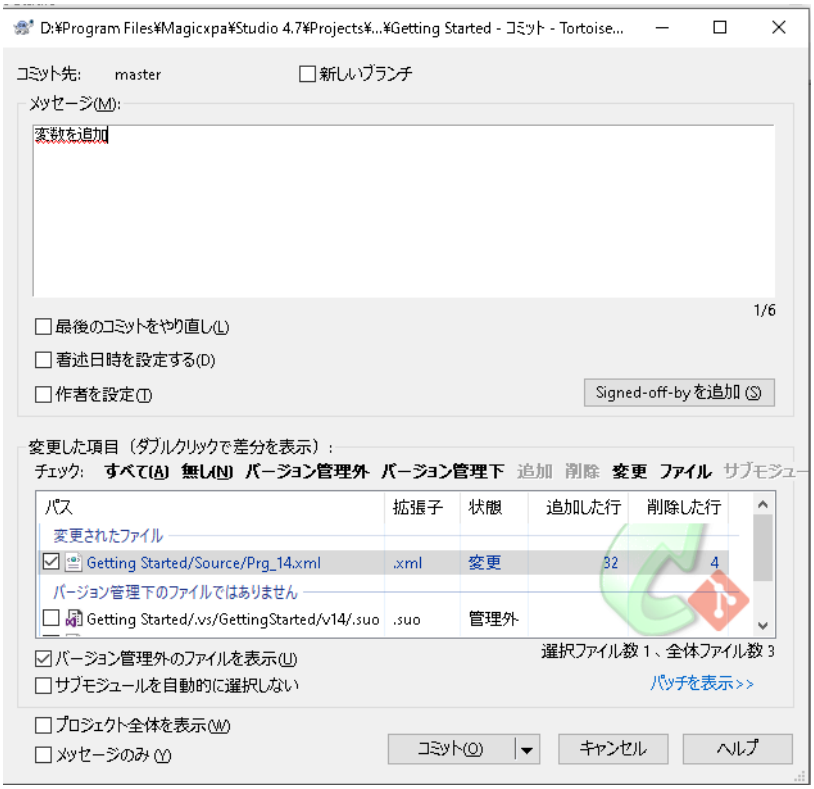


コミット

変更をコミットすると、作業ツリーに変更内容がローカルに保存されます。競合がない場合は、以下のように変更をコミットできます。

1. コミットする Magic xpa プログラムを選択します。
2. ［ファイル］メニューから［バージョン管理］>［コミット］を選択します。
3. コミットのコメントを入力します。ダイアログには、変更されたファイルが表示されます。

4. ［コミット］をクリックします。

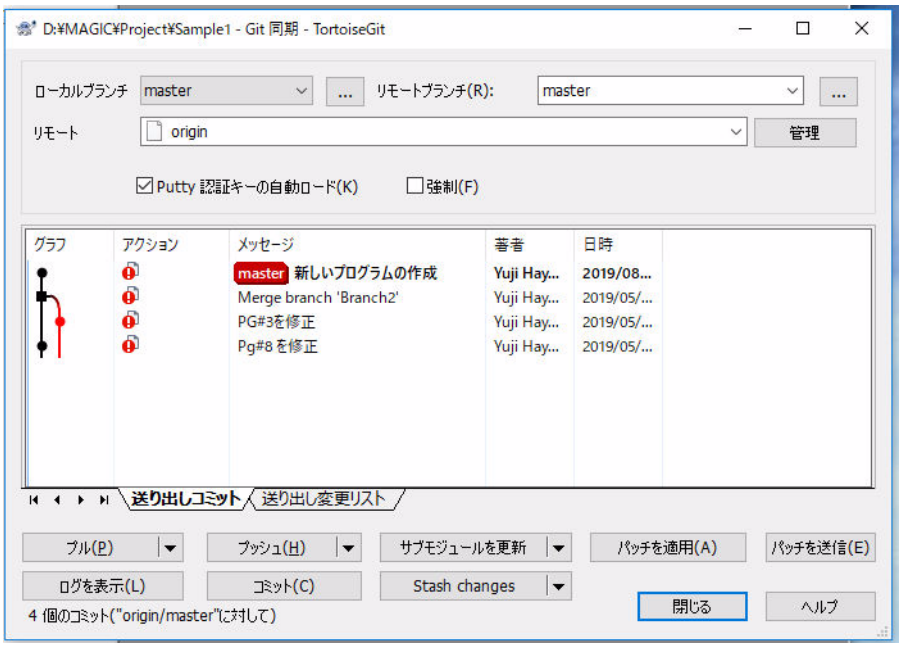


TortoiseGit でリモート処理を実行する

変更内容に対して Git リポジトリからのプルまたはプッシュ操作を実行することができます。

プッシュ / プル

プッシュ操作を使用すると、リモートサーバで新しいローカルコミットを公開することができます。プッシュは、ローカルリポジトリの内容をリモートリポジトリにアップロードします。プッシュ / プル操作を提供する［Git 同期］ダイアログボックス（バージョン管理 > Git 同期）でローカルリポジトリとリモート尾リポジトリを同期することができます。

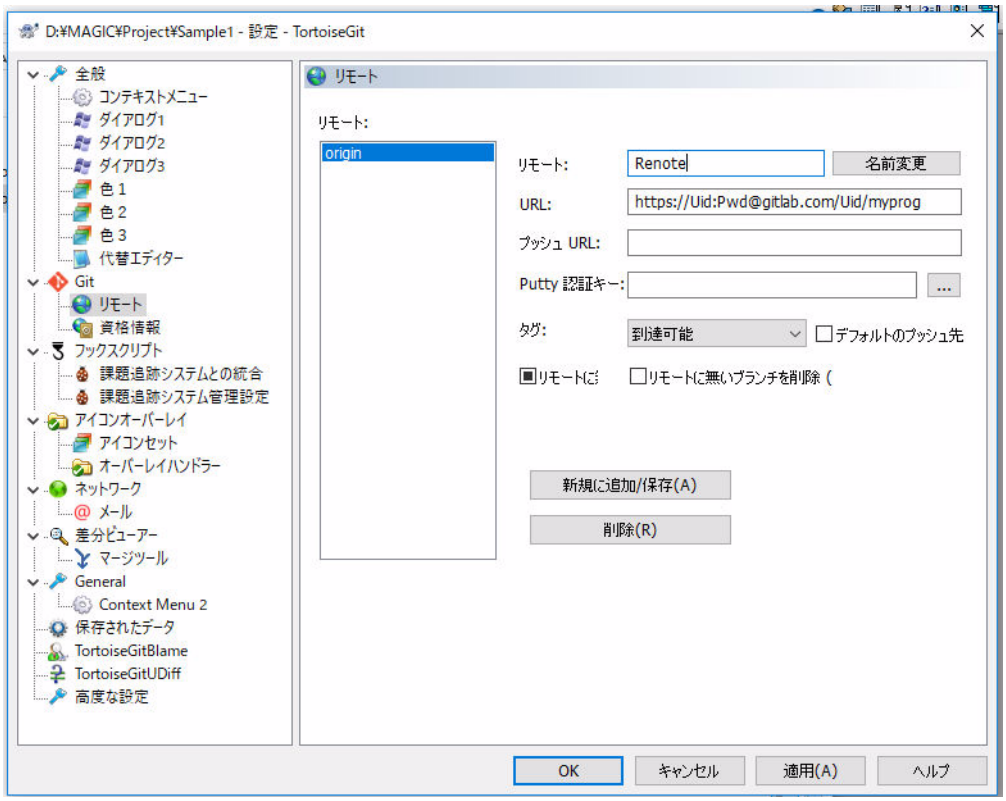


詳細は、次のサイトにアクセスしてください。https://tortoisegit.org/docs/tortoisegit/tgit-dug-pull.html.

URL を管理

TortoiseGit のリモート URL は次のように設定することができます。

1. プロジェクトフォルダ上で右クリックしてコンテキストメニューを開きます。［TortorioGit］>［設定］を選択して［設定］ダイアログを開きます。
2. Git >> リモート を選択します。
3. ［リモート］にリモートの名前に設定します（通常、デフォルトは「origin」と呼ばれます）。
4. ［URL］にローカルファイルシステムの URL に設定します。
5. 同じ URL をプッシュしたくない場合は、［プッシュ URL］に別の URL に設定します。同じ URL が必要な場合は、このフィールドを空白にしてください。
6. ［OK］をクリックします。



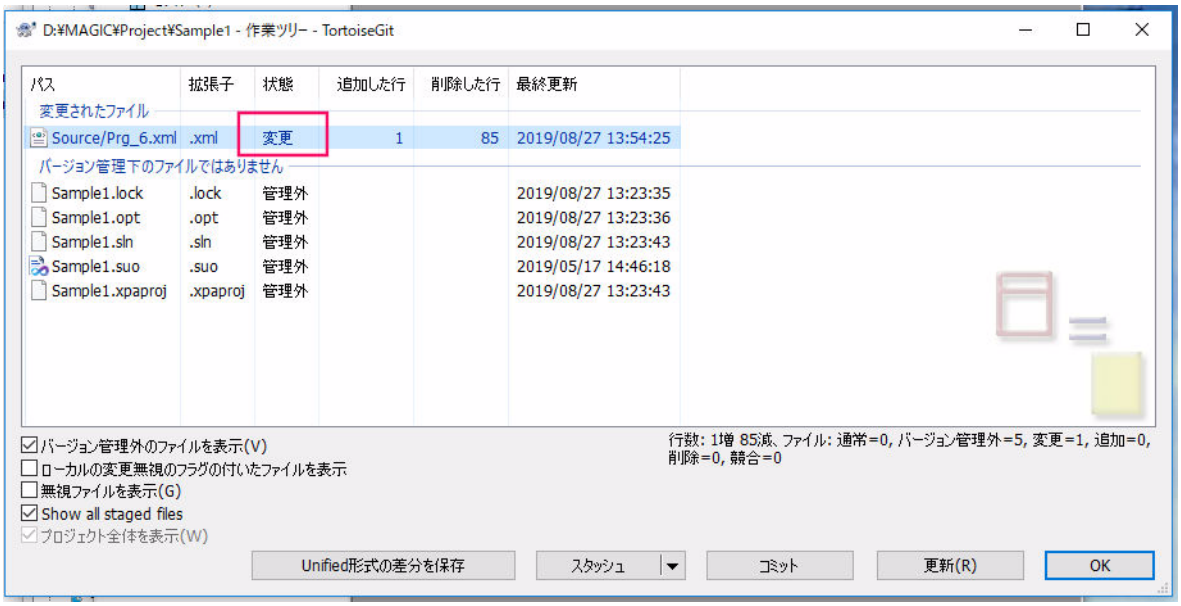
詳細は、次のサイトにアクセスしてください。https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html.

TortoiseGit を使用したその他の処理

次のようないくつかの処理があり、ローカルでもリモートでも実行することができます。

差分

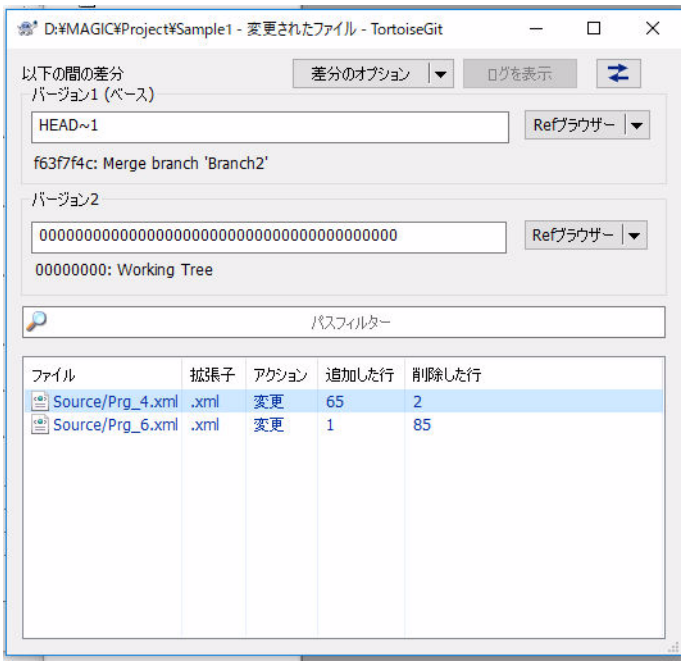
TortoiseGit は、プロジェクトが変更された時の差分をチェックします。差分を使用すると、作業ツリーで行ったコミットされていない変更を確認することができます。



差分の詳細については、次のサイトを参照してください。https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html.

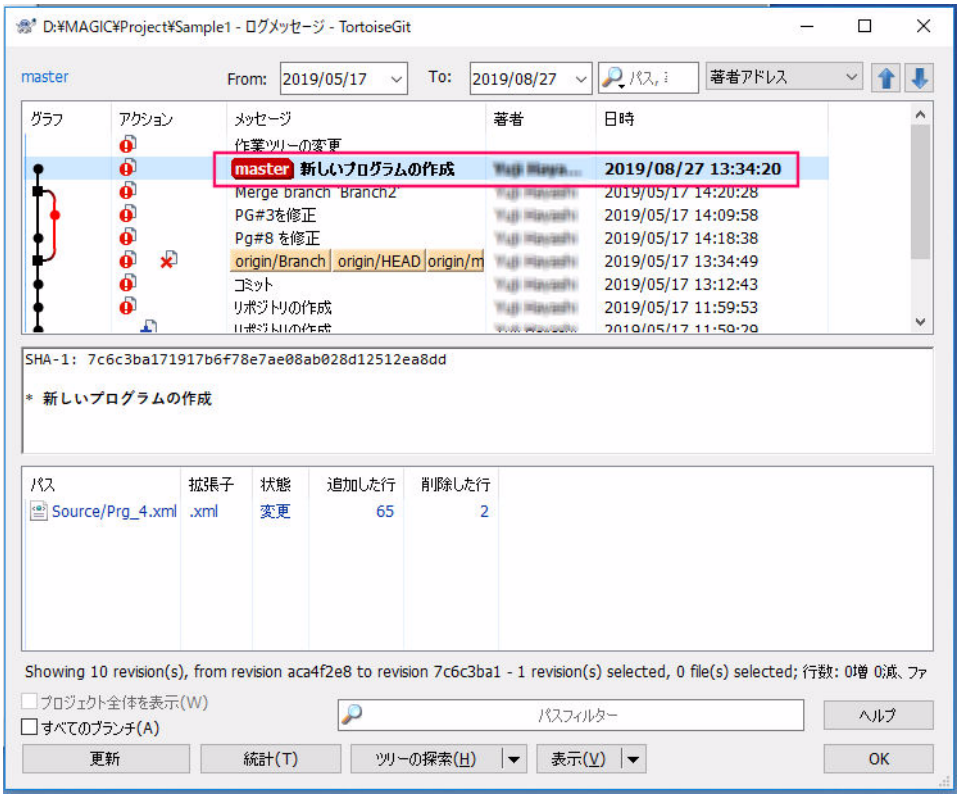
前のバージョンとの差分

ここでは、最後にコミットされたリビジョンと作業ツリーの差分を確認することができます。



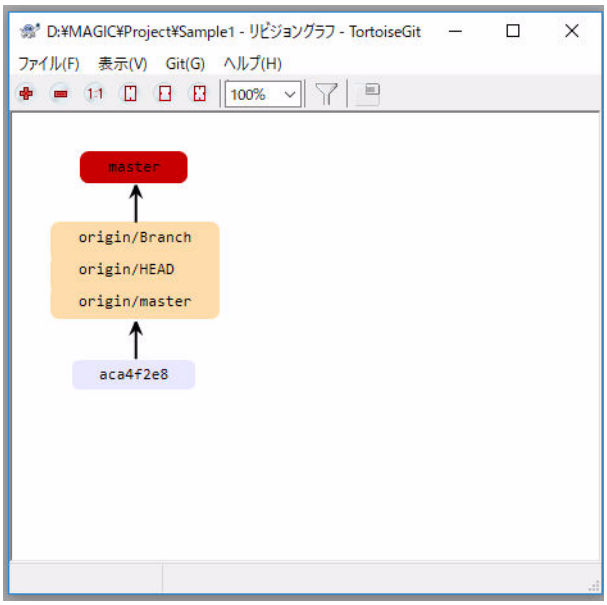
ログを表示

このオプションは、リポジトリに加えられた変更の履歴を開きます。



リビジョングラフ

TortoiseGit のリビジョングラフは、改訂履歴を分析し、タグ、ブランチ、およびその他の参照のノードを示す画像グラフを作成します。リビジョングラフの各ノードは、リポジトリの変更を表します。



レビジョンボックスにカーソルを合わせると、改訂日付、作成者、コメントがヒントボックスに表示されます。TortoiseGit の一般設定でグラフの外観を設定することができます。

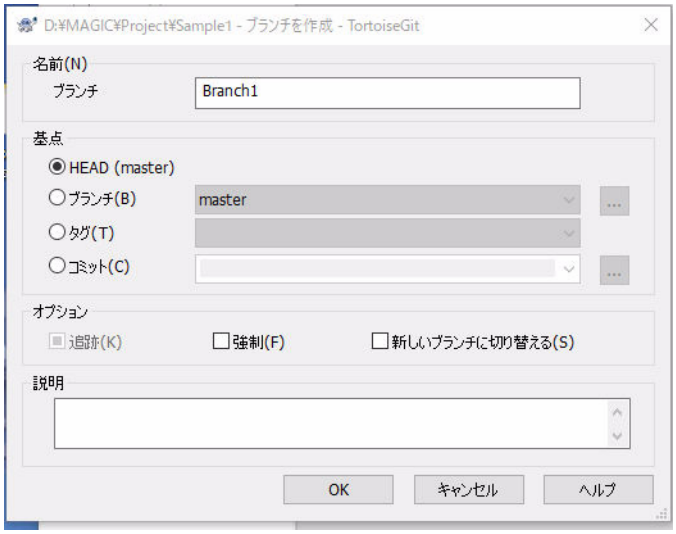
分岐とマージ

バージョン管理システムの顕著な特徴の 1 つは、変更を別の開発ラインに分岐できることです。この行はブランチと呼ばれます。最終的に作業ツリーを目的のブランチにマージすることができます。

ブランチを作成

分岐は、開発のメインラインを妨げることなく新しい機能を試すために作成されます。以下のようにブランチを作成することができます。

1. [ファイル］メニューから［バージョン管理］>［ブランチの作成］を選択します。
2. ブランチに名前を付けます。
3. ［OK］をクリックします。



ダイアログボックスの他のオプションの詳細は、次のサイトを参照してください。https://tortoisegit.org/docs/tortoisegit/tgit-dug-branchtag.html.

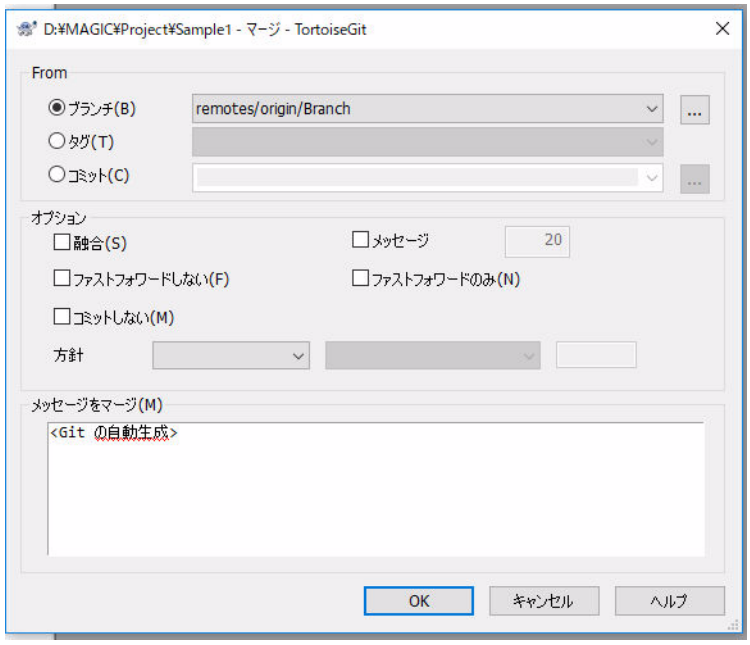
ブランチをマージする

開発中のブランチがある場合、その後、ブランチで行っていた変更が確定します。その時点で、実績のあるブランチにマージする必要があります。

以下のようにブランチを安定ブランチにマージすることができます。

1. ［ファイル］メニューから［バージョン管理］>［マージ］を選択します。
2. マージするブランチを選択します。
3. ［OK］をクリックします。

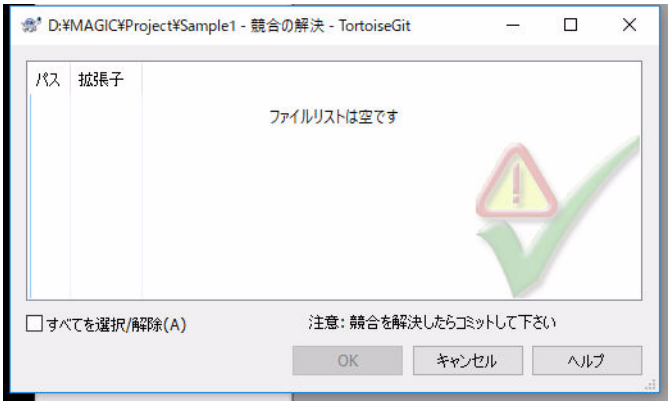
マージは常に作業ツリー内で行われます。変更を確立されたブランチにマージする必要がある場合、そのブランチの作業ツリーをチェックアウトし、次のようにマージを呼び出す必要があります。



競合の解決

変更が自動的にマージされない場合があります。これは競合する状況によるものです。次のようにして、個々のファイルの競合を解決することができます。

1. ［バージョン管理］>［競合の解決］を選択します。競合するファイルは赤で強調表示されます。

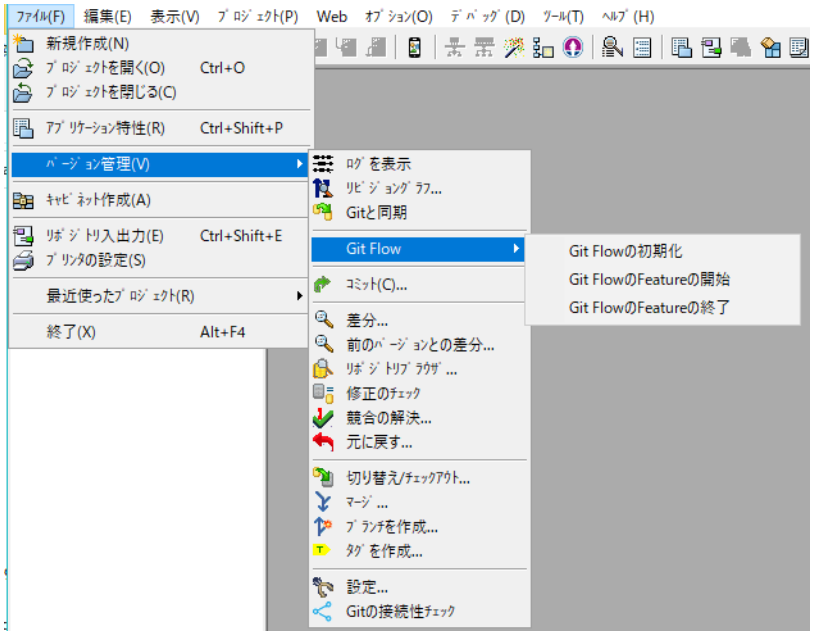


2. 競合するファイルで右クリックします。
3. 解決のオプションのいずれかを選択します。

Magic xpa から Git flow コマンドを使用する

Git flow とは、Git におけるリポジトリのブランチモデルの導入を簡単にするための Git プラグインです。

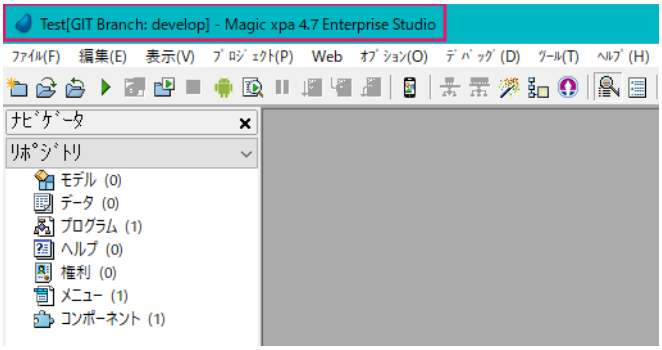
Git Flow は、バージョンリリースを中心に設計された厳密なブランチモデルを定義しています。Git の機能は通常、開発者のリポジトリに存在します。Gitf Fow コマンドをサポートするために、Magic xpa は以下のように［バージョン管理］メニューの下に［Git Flow］という新しいオプションが追加されました。



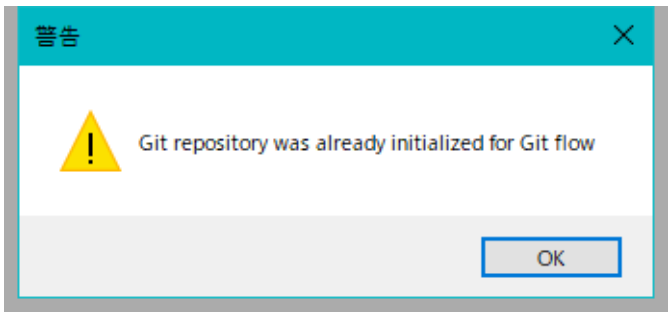
[ファイル > バージョン管理] メニューの [Git Flow] エントリには、以下のオプションが用意されています。

Git Flow の初期化

このコマンドは、ディレクトリを Git Flow リポジトリとして初期化します。これによって "develop" ブランチが作成されます。



すでに初期化されている場合、Magic xpa はエラーを表示します。



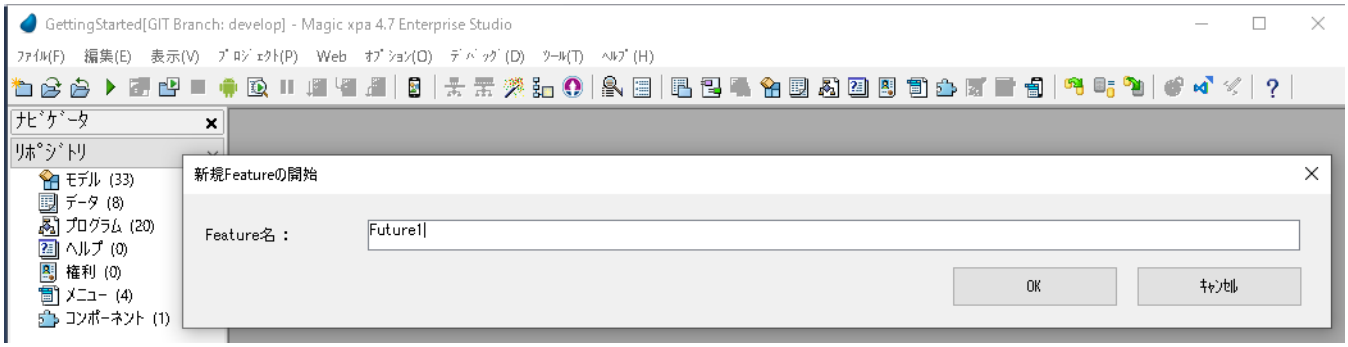
次のコマンドを実行します。

```
git.exe flow init -d
```

この後、[ファイル > バージョン管理] メニューの [切り替え / チェックアウト] を選択して、"develop" ブランチに切り替えます。

Git Flow の Future の開始

このコマンドは、Git Flow の Feature ブランチを開始します。これによって、ローカルリポジトリに Feature ブランチが作成され、特定の機能を開発するためにブランチが切り替わります。



次のコマンドを実行します。

```
git.exe flow feature star <Futute 名 >
```

Git Flow の Future の終了

機能の開発作業が完了したら、このコマンドを使用して Feature ブランチを閉じることができます。

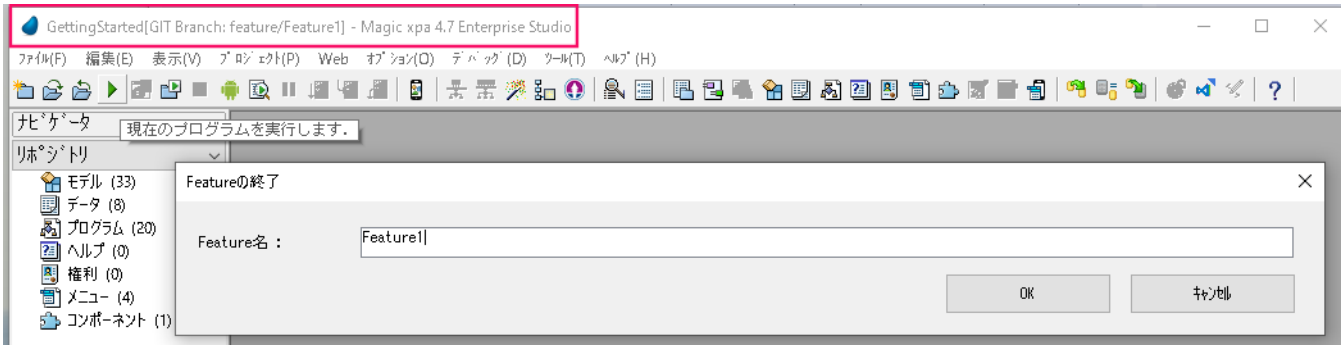
終了処理は以下を自動的に行います。

1. "develop" ブランチに切り替えます。
2. Feature ブランチを "develop" ブランチにマージします。
3. ［Git Flow の Feature を終了］を選択して Feature ブランチを削除します。

ノート

予め修正内容をコミットしておいてください。

開始時に作成した Feature ブランチ と同じ名前を指定してください。



次のコマンドを実行します。

```
git.exe flow feature finish <Futute 名 >
```

Git Flow の機能開発が終了すると、その Feature ブランチはメインの Develop リポジトリにマージされます。

Git Flow の詳細については、https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow などを参照してください。

Magic xpa の Git 機能を無効化する

プロジェクトがすでに Git の配下であっても、Git を利用しないプロジェクトとして扱うことができます。

"SpecialDisableGITFunctionality" という特殊フラグが利用できるようになりました。

MAGIC.INI の [MAGIC_SPECIALS] セクションに以下のように設定します。

```
SpecialDisableGITFunctionality = Y
```

これによって GIT によるバージョン管理として追加されたプロジェクトであっても、すべての GIT 関連のアイコンや［ファイル > バージョン管理］内ののすべてのメニューオプションが無効になります。

このフラグは GIT プロジェクトを非 Git プロジェクトとして扱い、オフラインでの作業を可能にします。

Magic xpa で Git が生成したコマンドをログに記録する

Magic xpa Studio から実行されたすべての Git コマンドをログに記録することができます。

"SpecialGITLog" という特殊フラグが利用できるようになりました。

MAGIC.INI の [MAGIC_SPECIALS] セクションに以下のように設定します。

```
SpecialGITLog = <full Filename>
```

これによって、指定したファイルにログが記録されるようになります。

<full Filename> はログファイルが作成される絶対パスを持つファイル名で、Magic xpa Studio で実行されたすべての Git コマンドがそのファイルに記録されます。

例：

```
[MAGIC_SPECIALS]SpecialGITLog = Gitlog¥myGitlog.log
```

上の例のようにファイルパスが相対パスの場合、ログファイルは作業ディレクトリに作成されます。

推奨：

ログファイルは、現在のローカルの Git リポジトリの外に置かなければなりません。開発者にとって、特にブランチを変更しているときにログの内容を見つけられるようにすることは非常に重要です。

必須事項：

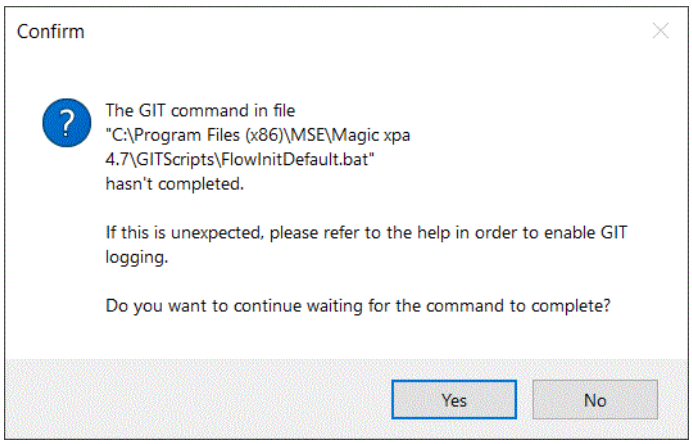
.gitignore でこのファイルを無視する必要があります。

Git コマンドの問題のトラブルシューティング

Git コマンドの実行時のデバッグをより明確にするために、Magic xpa では Git を直接呼び出すのではなく、バッチファイルを実行し、そのバッチファイルがそれぞれの Git コマンドを実行します。

- バッチファイルは Magic xpa のインストール時に作成された "GitScripts" フォルダに格納されます。
- それぞれのコマンドには個別のバッチファイルが用意されています。

バッチファイルは、どのステップ / コマンドで問題が発生したのか、そのコマンドの実行に本当に時間がかかっているのか、それともそのコマンドが単にマシンの進行を妨げているだけなのかを特定する場合に役立ちます。Git コマンドの実行に 15 秒以上かかっている場合（この時間はカスタマイズできません）、Magic xpa は確認ウィンドウを表示して、それ以上待つか、現在の処理を停止するかを判断します。確認ウィンドウは以下のようになります。



- 「Yes」をクリックすると、Magic xpa は Git コマンドの実行が完了するまでさらに 15 秒待ちます。
- 「No」をクリックすると、Magic xpa は現在のコマンドの実行を停止します。

Git コマンドの実行中のエラーは、[MAGIC_SPECIALS]SpecialGITLog = GitlogGitlog\myGitlog.log で定義されたログファイルに記録されます。

Magic xpa はバッチファイルの中に 2 つのパラメータを用意しています。これにより、トラブルシューティングやデバッグ、Git で何が起こっているかをチェックすることができます。デフォルトでは、これら二つのパラメータはマークされているので、パラメータを使用するには 'rem' を削除する必要があります。

```
set GIT_TRACE=true

set GIT_CURL_VERBOSE=true
```

これらのパラメータは、Git の操作の詳細なメッセージを表示します。両方を一緒に有効にする必要があります。

GitScripts フォルダの中に GitScripts.zip というファイルがあります。バッチファイルに何か変更があった場合は、このファイルを解凍して元のバージョンのバッチファイルを取得することができます。

Git Hub の使い方

GitHub とは

GitHub とは、ソフトウェア開発プロジェクトのためのソースコード管理サービスです。

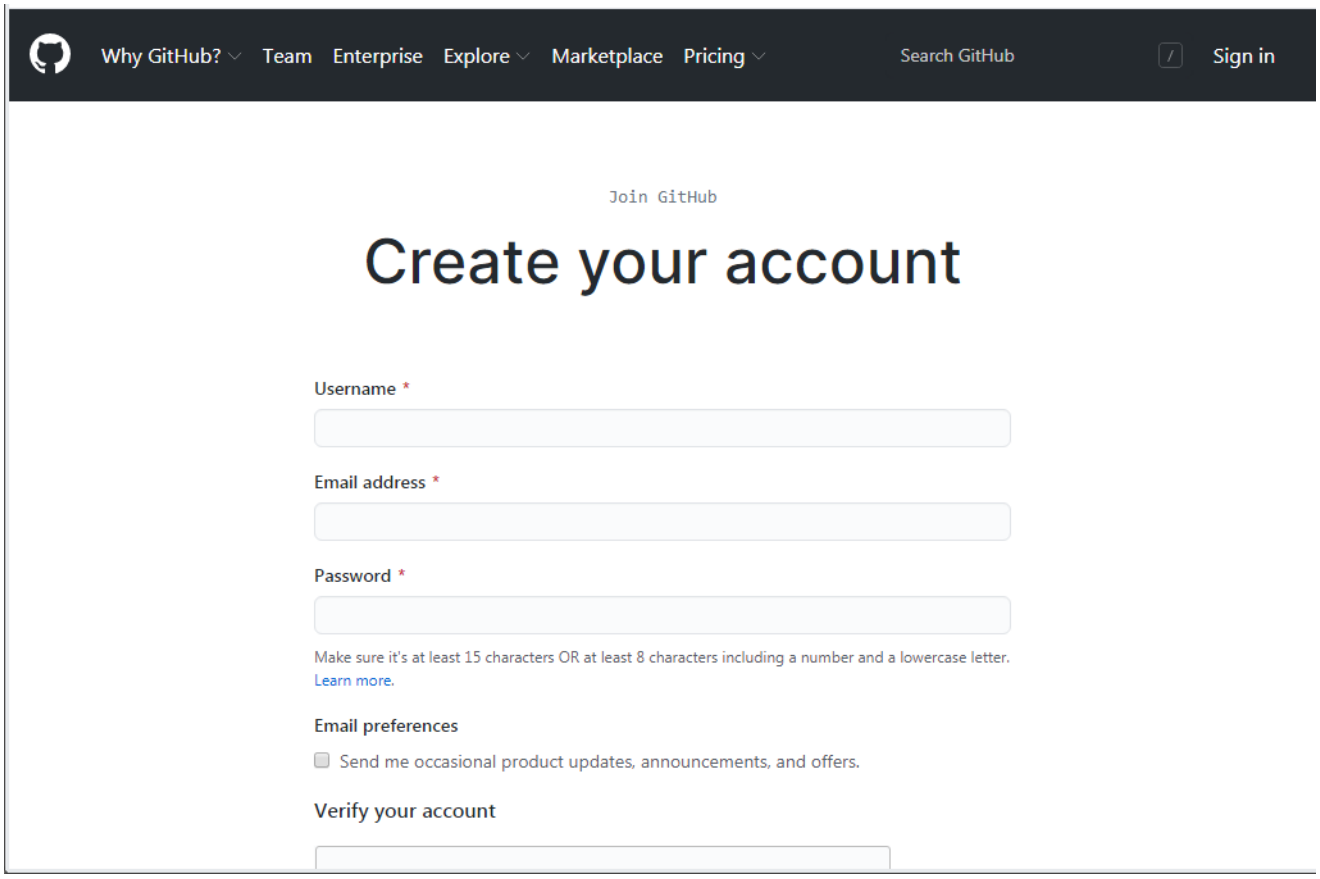
公開されているソースコードの閲覧や簡単なバグ管理機能、SNS の機能を備えており、開発者にとって無くてはならないサービスです。

また、GitHub を使ってバージョン管理を行っている企業も多数あります。

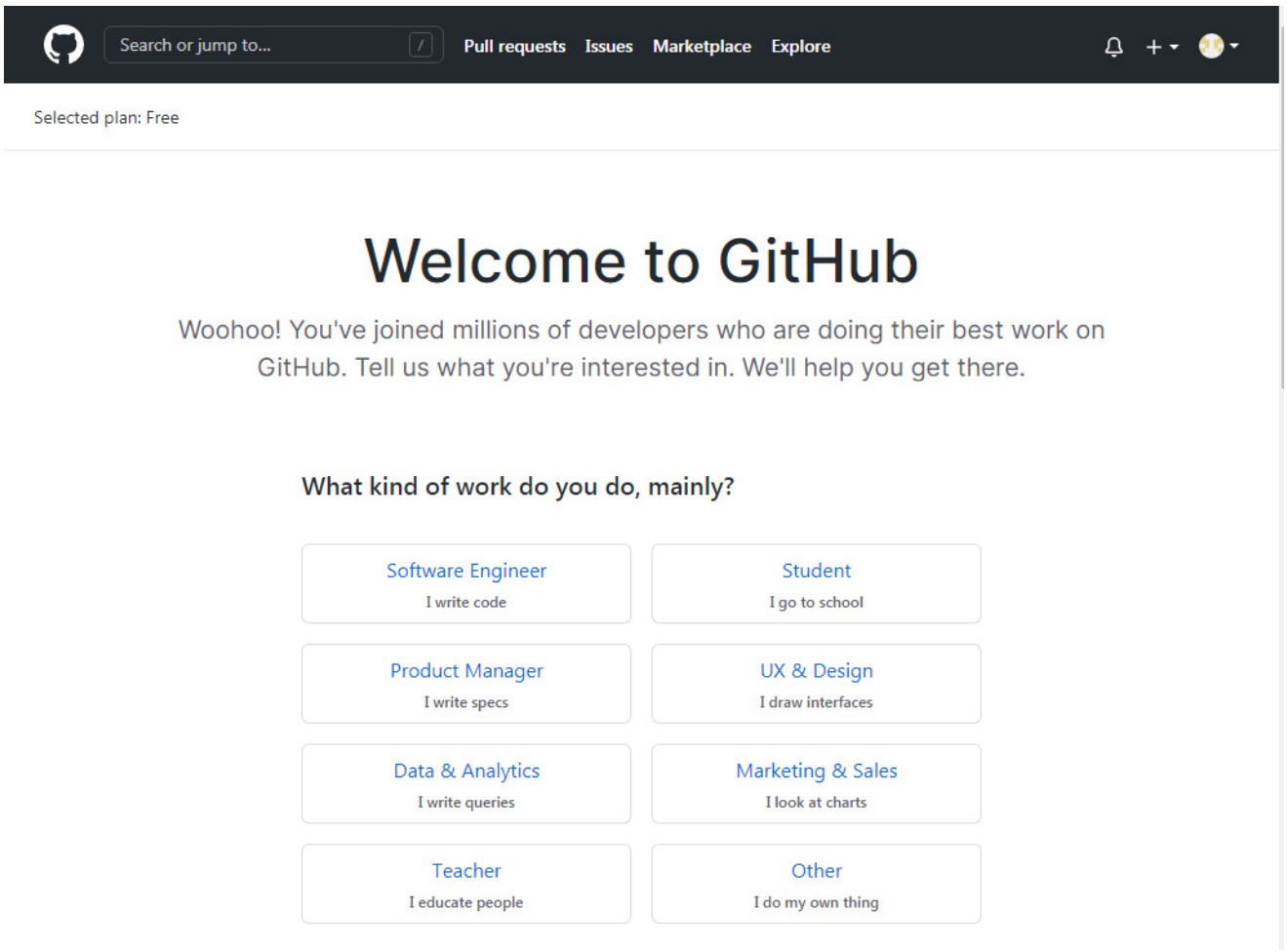
GitHub のアカウント登録

GitHub に登録してみましょう。まずは、GitHub のトップページにアクセスします。

ここで、ユーザ名とメールアドレス、パスワードを入力して、アカウント登録を行ってください。



続いて、業務内容やプログラミング経験、Git ub の利用目的などを選択する画面が表示されます。



登録したメールアドレスに認証のメールが届きます。メールの内容に従いユーザ認証を行ってください。

以上で GitHub のアカウント登録は完了です。

GitHub を使う上で知っておきたい事前知識

使い始める前に知っておきたい事前知識を 3 つ紹介します。

ここで紹介する言葉がまったくわからない場合は、理解しておきましょう。

ローカルリポジトリとリモートリポジトリ

リポジトリとは、ファイルやディレクトリの状態を保存する場所です。変更履歴を管理したいディレクトリなどをリポジトリの管理下に置くことで、そのディレクトリ内のファイルなどの変更履歴を記録することができます。

リポジトリは自分のマシン内にある「ローカルリポジトリ」とサーバなどネットワーク上にある「リモートリポジトリ」の 2 箇所にあります。基本的にローカルリポジトリで作業を行い、その作業内容をリモートポジトリへプッシュする流れで行います。

コミットとプッシュ

- コミット (commit) ：ファイルの追加や変更の履歴をリポジトリに保存すること
- プッシュ (push) ：ファイルの追加や変更の履歴をリモートリポジトリにアップロードするための操作

ブランチ（branch）

ソフトウェアの開発では、現在リリースしてるバージョンのメンテナンスをしながら新たな機能追加やバグ修正を行うことがあります。このような、並行して行われる複数のバージョン管理を行うために、Git にはブランチ（branch）という機能があります。

ブランチは履歴の流れを分岐して記録していくものです。分岐したブランチは他のブランチの影響を受けないため、同じリポジトリ内でそれぞれの開発を行っていくことができます。

GitHub の使い方

基本的な Git の作業は下記のような手順になります。#1 の作成は初回のみ行い、#2 から #5 を繰り返します。

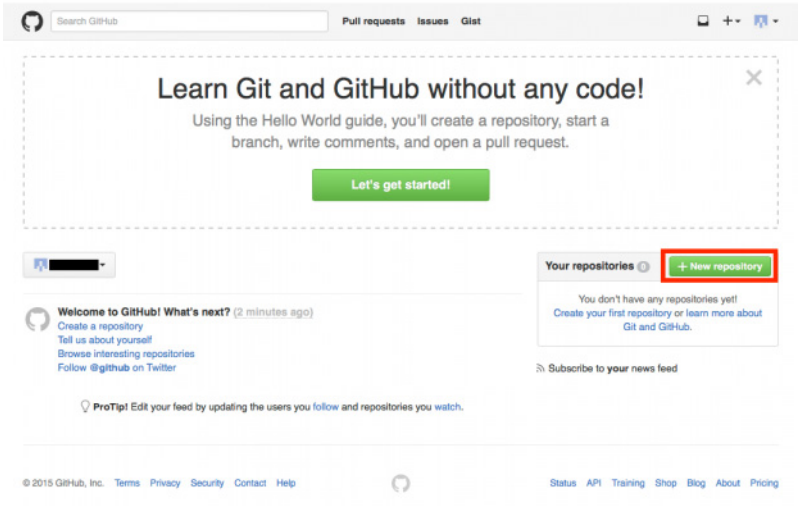
基本的に小さい作業の単位でコミットを行い、ある程度作業がひと段落した時にプッシュをするのが一般的です。コミットの作業がわかりやすいように、コミットメッセージを残しておくと、ログを追っていく時に役立ちますので覚えておきましょう。

1. Github でリポジトリを作成（git init）、または複製（git clone）する
2. ファイルの作成、編集を行う
3. ファイルの作成 / 変更 / 削除を git のインデックスに追加する（git add）
4. 変更結果をローカルリポジトリにコミットする（git commit）
5. ローカルリポジトリをプッシュしてリモートリポジトリへ反映させる（git push）

GitHub でリポジトリを作成する

まずはリポジトリを作成します。

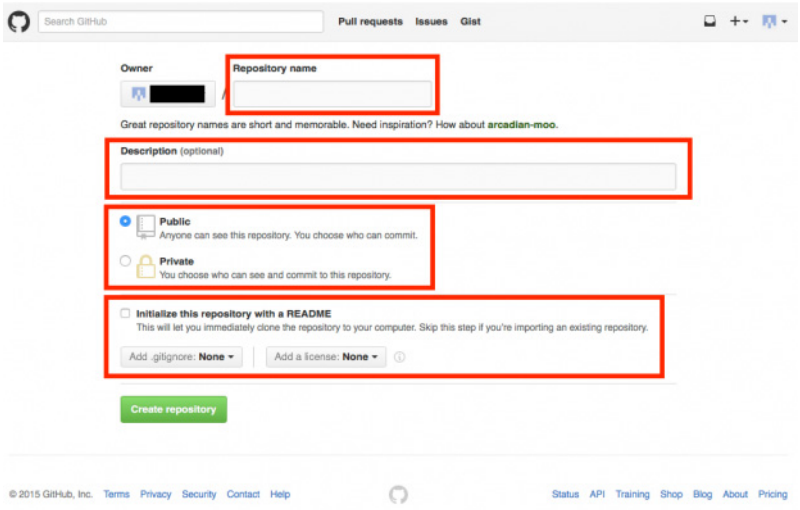
GitHub にログインした状態で、「New Repository」ボタンを押下します。



次に表示される画面では、「Repository name」の入力のあと、必要に応じて「Description」も入力します。

また、リポジトリの種類を「Public」か「Private」を選択します。「Private」リポジトリは、有料会員のみ作成することが可能です。

最後に、リポジトリの中にあらかじめ README ファイルを作成しておく場合は「Initialize this repository with a README」にチェックを入れます。.gitignore や license については後で追加や変更ができますので、None を選択します。



必要項目の入力が終わり「Create repository」ボタンをクリックするとリポジトリの作成は完了です。

次の画面で、リモートリポジトリのアドレスが表示されますので、控えておいてください。

ファイルの作成、編集を行う

「hello.html」というファイルをローカルの PC 上に作成した想定で進めます。

テキストエディタなどで HTML ファイルを作ります。

はじめに、ローカルの PC 上にローカルリポジトリを作成します。

今回は「awesome」というディレクトリを作成することにします。

```
mkdir awesome
cd awesome
git init
```

awesome というディレクトリを作成し、そのディレクトリに移動してから作業が始まります。

「git init」コマンドは Git リポジトリを新たに作成するコマンドです。

バージョン管理を行っていない既存のプロジェクトを Git リポジトリに変換する場合や、空の新規リポジトリを作成して初期化する場合に使用します。git init コマンドを実行するとカレントディレクトリを Git リポジトリに変換します。

ファイルの作成 / 変更 / 削除を git のインデックスに追加する（git add）

先ほど作成した「hello.html」のファイルをローカルリポジトリに追加しましょう。

以下のコマンドでインデックスに追加します。

インデックスとは、リポジトリにコミットする準備をするために変更内容を一時的に保存する場所のことです。

```
git add hello.html
```

変更結果をローカルリポジトリにコミットする（git commit）

次に、インデックスに追加されたファイルをコミットします。

コミットとは、ファイルやディレクトリの追加や変更をリポジトリに記録する操作のことです。

```
git commit -m "add new file"
```

これで、リポジトリに対してファイルの追加が記録されました。

ファイルが追加されているか確認します。

```
git status
```

さらに、リモートリポジトリに反映させる前に、リモートリポジトリの情報を追加します。この情報は、先ほど GitHub 上に表示された、リモートリポジトリのアドレスです。今回は例を示します。

```
git remote add origin https://github.com//awesome.git
```

ローカルリポジトリをプッシュしてリモートリポジトリへ反映させる（git push）

ローカルリポジトリの変更を、GitHub 上にあるリモートリポジトリに反映させるため、以下のコマンドを実行します。

```
git push origin master
```

GitHub のユーザ名とパスワードを尋ねられますので入力してください。

これで、GitHub へプッシュしてリモートリポジトリへ反映させることができました。