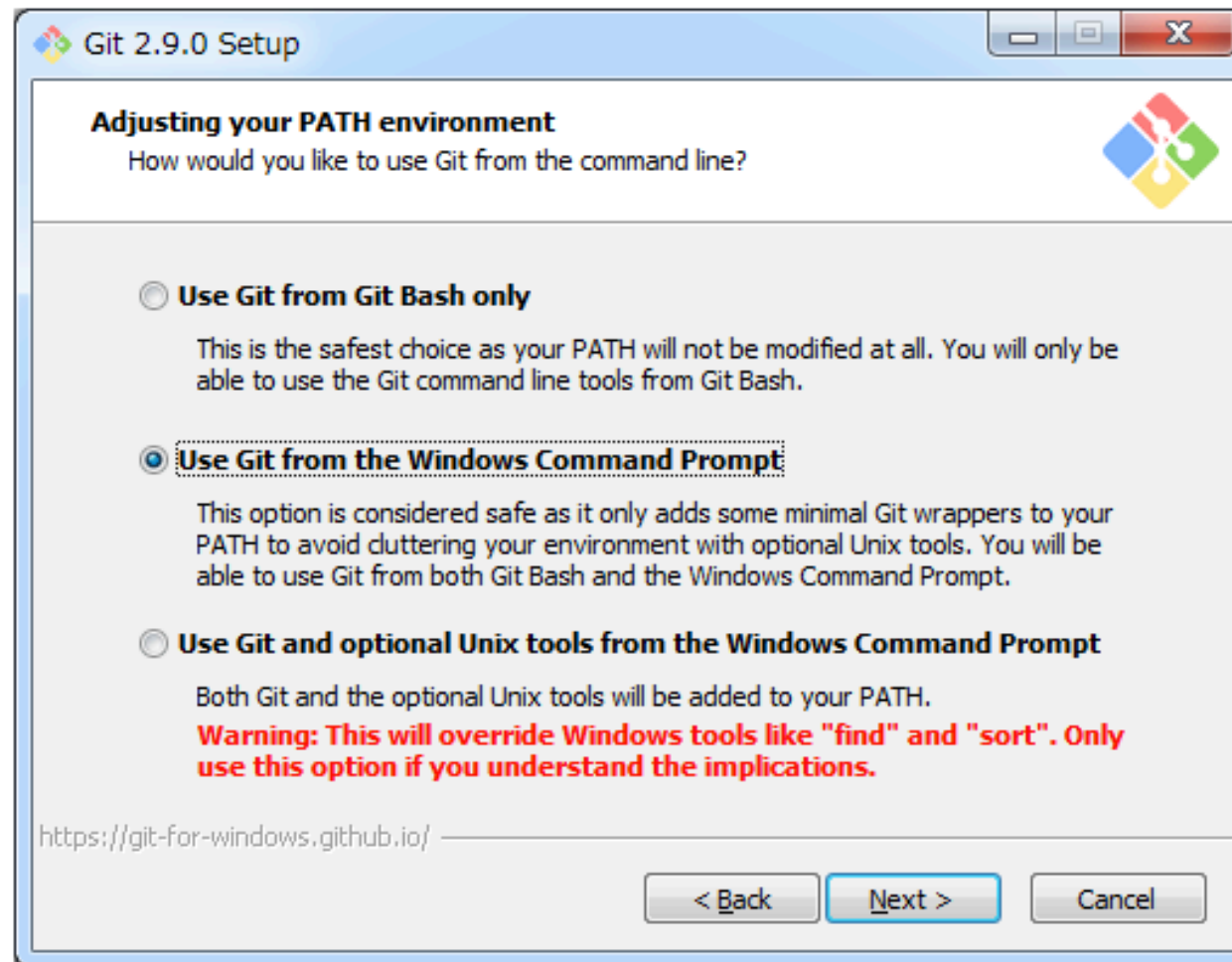


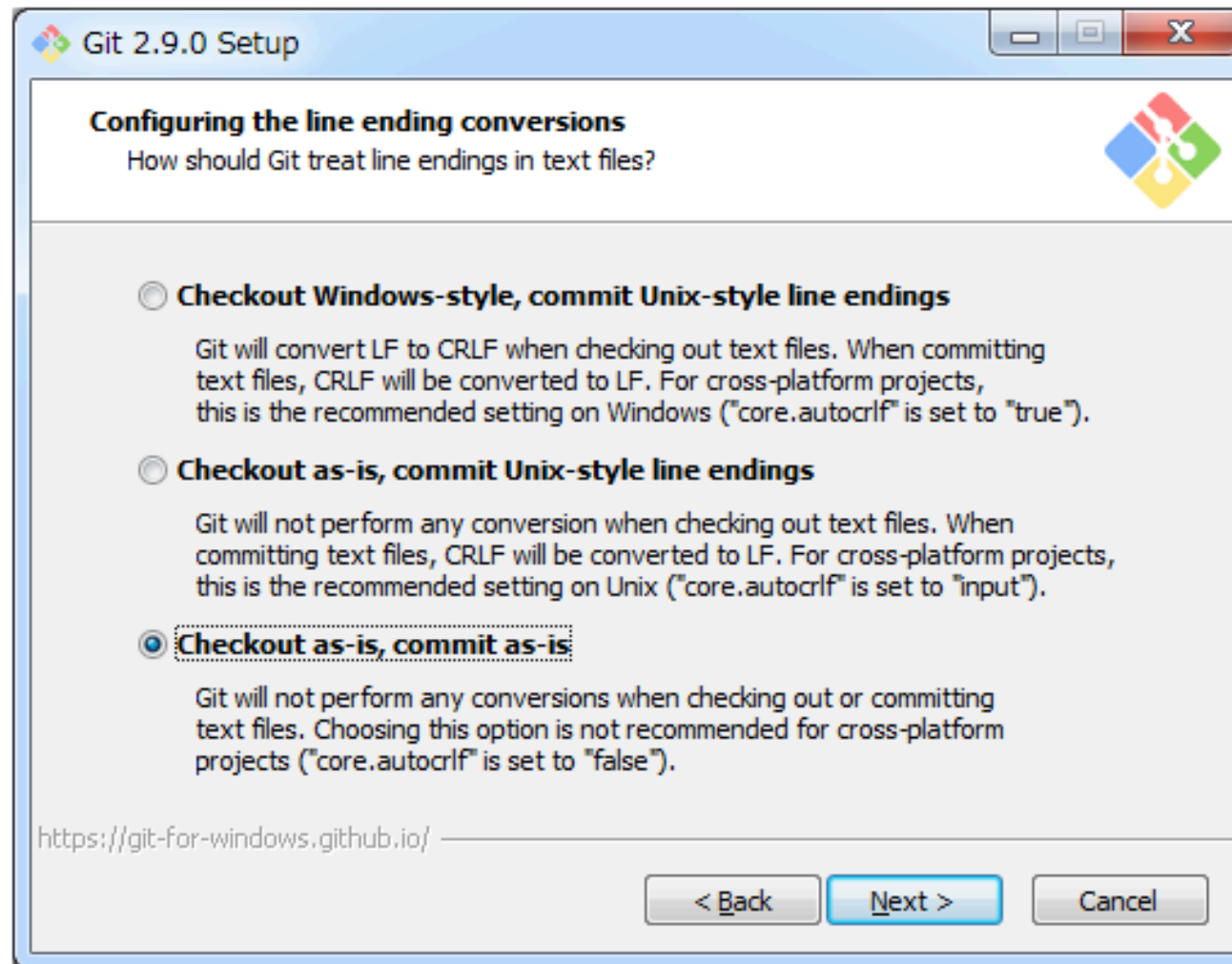
Gitのインストール

Gitの公式サイト（git-scm.com）からGitをダウンロードしてインストールします



ここは2番めを選択します（おそらくデフォルト）

Gitのインストール



ここは3番めを選択します（とても重要）

他の設定は、デフォルトか任意の設定で大丈夫です

GitLabとの連携

コマンドプロンプトを開きます

`git -version` で、正しくインストールされているかを確認します

初期設定をします

`git config -global user.name "ユーザ名"` GitLabに登録したもの

`git config --global user.email "メールアドレス"` GitLabに登録したもの

`git config --global core.quotepath false` 日本語をエスケープしない設定

上記の設定はホームディレクトリの.gitconfigファイルに定義されています

`cd ~`
`ls -la | grep .gitconfig` .gitconfigが生成されているかを確認

GitLabとの連携

ホームディレクトリに.sshディレクトリを生成します

```
mkdir ~/.ssh  
cd ~/.ssh
```

SSH通信に使用する鍵を生成します

```
ssh-keygen -t rsa -C “メールアドレス”
```

Keyを作成するか聞かれるのでEnter

パスフレーズを入力、再入力（後で使用するので忘れないように）

.sshディレクトリ内にrsa.pubファイルが生成されるので、
テキストエディタで開き、中身を全コピーします

GitLabとの連携

GitLabに戻って、右上のプロフィールアイコンから、設定を選択します

ユーザー設定 > SSH 鍵

SSH 鍵

SSH 鍵を使用すると、コンピュータとGitLabの間を安全に接続できます。

SSH 鍵を追加

SSH キーを追加するには、[キーを生成する](#) または [既存のキーを使用する](#) のどちらかが必要です。

キー

あなたの SSH 公開鍵を貼り付けます。この鍵は通常、ファイル `~/.ssh/id_ed25519.pub` または `~/.ssh/id_rsa.pub` 内にあり、`'ssh-ed25519'` または `'ssh-rsa'` という文字列から始まります。SSH の秘密鍵を使用しないようにしてください。

典型的には `"ssh-ed25519 ..."` または `"ssh-rsa ..."` で始まります

タイトル	Expires at
<input type="text" value="例:MacBook のキー"/>	<input type="text" value="年 / 月 / 日"/>

Give your individual key a title. This will be publically visible.

サイドメニューからSSH鍵を開き、先程コピーした鍵データを貼り付けます

タイトルを適当に入力してキーを追加を押します

(複数PCを使っている場合は、PCごとに設定が必要なので、
タイトルにPC名を入れておくと管理が楽です)

用語の説明（リモートとローカルの違い）

リモートはサーバ上のソース（現状で言う**FTP**接続先）

ローカルは作業マシン上のソース（自分の**PC**内）

用語の説明（コマンドについて）

checkout . . . 作業ブランチを切り替えるコマンド
（チェックアウト） 新しい作業ブランチを作ることができる

add . . . ソースファイルをGit管理下を含めるコマンド
（アド） 新しいファイルを生成したときに使う

Commit . . . ソースファイルへの変更を確定させるコマンド
（コミット） これをしないとマージやプッシュができない

Merge . . . ソースファイルを合体させるコマンド
（マージ） 複数人がファイルを修正していてもよしなに結合してくれる

pull . . . リモートからローカルにソースを引っ張ってくる
（プル） 基本はリモート**MASTER**からローカル**MASTER**へ行うコマンド
FTPのダウンロードみたいなもの

push . . . ローカルからリモートへファイルを送信するコマンド
（プッシュ） FTPのアップロードみたいなもの

用語の説明（登場する名称について）

リモートMASTERブランチ	テストサーバ上に公開されるファイル群 常にこれを正とする
ローカルMASTERブランチ	リモートMASTERブランチのコピー ローカルでGit作業をする前にコピーする 現状ではバックアップを取る行為にあたる
ローカル作業ブランチ	実作業を行うブランチ 課題や単位で作成して作業を行う

Git使用の流れ①

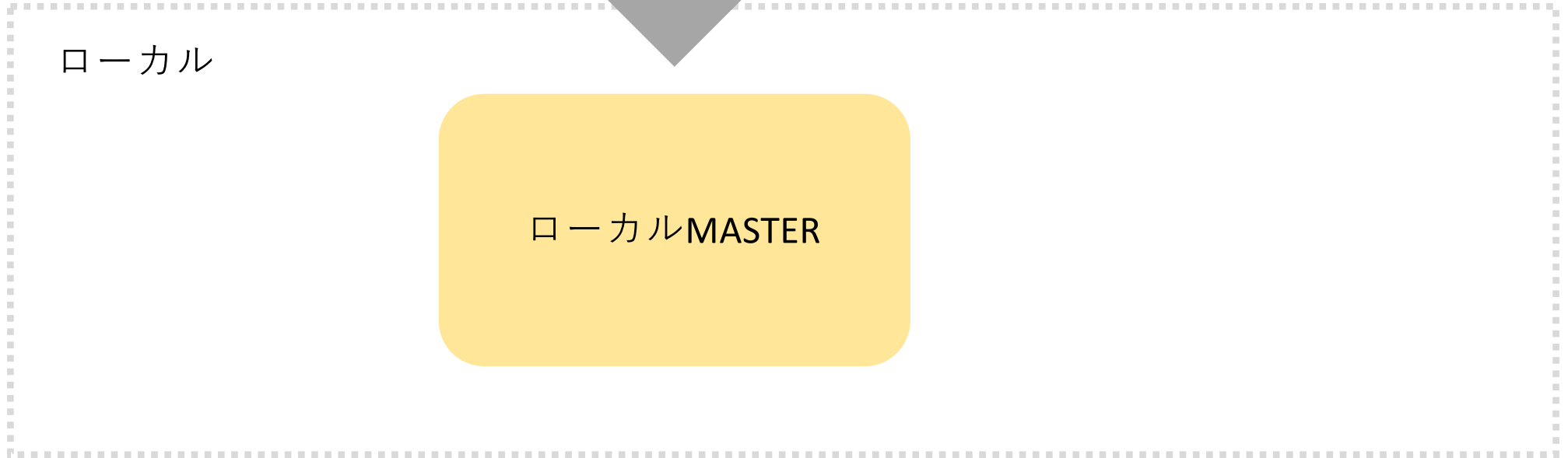
リモートのMASTERを
ローカルにコピーします

リモート MASTER

clone



ローカル

ローカル MASTER



GitLabの駅前不動産のページに行きます


ktt > ... > external > prj-ekimae > 詳細

 **prj-ekimae** 
プロジェクトID: 19241598

  Star 0  フォーク 0

🔗 25 コミット 🌿 4 ブランチ 🏷️ 0 タグ 📁 73.4MB ファイル 🗄️ 76.3MB Storage

駅前不動産サイト制作




Auto DevOps

あらかじめ定義された CI/CD の構成を基に自動的にアプリケーションをビルド、テスト、デプロイします。

詳しくは、 [Auto DevOps ドキュメント](#) をご覧ください。

設定を有効にする



master ▼ prj-ekimae / + ▼ 履歴 ファイルを検索 Web IDE 📄 ▼ クローン ▼



Merge branch '15-建物・部屋へのページ遷移方法をGETへ変更' into 'master' ...
koyanagi-ktt authored 7 hours ago

44b1e11d 

-  README を追加
-  ライセンスの追加
-  変更履歴を追加
-  CONTRIBUTINGを追加
-  Kubernetes クラスターを追加
-  CI/CD を設定

名前	最新コミット	最終更新
📁 articles	save	2 weeks ago

クローンを押して、SSHでクローンのコマンドをコピーして
コマンドプロンプトにて実行します

※今いる場所にディレクトリが生成されるので、注意してください

ktt > ... > external > prj-ekimae > 詳細

prj-ekimae プロジェクトID: 19241598 🔔 ☆ Star 0 🍴 フォーク 0

🔗 25 コミット 🌿 4 ブランチ 🏷 0 タグ 📁 73.4MB ファイル 🗄 76.3MB Storage

駅前不動産サイト制作

Auto DevOps
あらかじめ定義された CI/CD の構成を基に自動的にアプリケーションをビルド、テスト、デプロイします。
詳しくは、[Auto DevOps ドキュメント](#) をご覧ください。
設定を有効にする

master ▾ prj-ekimae / + ▾ 履歴 ファイルを検索 Web IDE 📄 **クローン ▾**

Merge branch '15-建物・部屋へのページ遷移方法をGETへ変更' into 'master' ...
koyanagi-ktt authored 7 hours ago

📖 README を追加 📄 ライセンスの追加 📅 変更履歴を追加 📄 CONTRIBUTINGを追加 🔧 CI/CD を設定

SSH でクローン
git@gitlab.com:ktt/wp/external:prj-ekimae

HTTPS でクローン
https://gitlab.com/ktt/wp/external:prj-ekimae

名前	最新コミット	最終更新
📁 articles	save	2 weeks ago

Git使用の流れ②

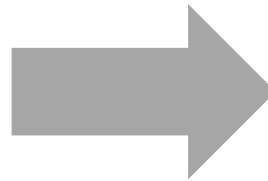
MASTERブランチをコピーして
作業ブランチを作成します

リモート MASTER

ローカル

ローカルMASTER

checkout -b



作業ブランチ

クローンしたディレクトリに移動し、作業ブランチを生成します

```
cd prj-ekimae
```

```
git checkout -b 作業ブランチ名
```

作業ブランチ名は
作業内容を完結にしたもので良いです

※補足説明

Checkoutコマンドは、本来はブランチ間を移動するためのコマンドです

```
git checkout 移動先ブランチ名
```

で、対象のブランチに移動できます

-b オプションをつけると、移動先のブランチが存在しないとき作成します

また、 `git branch` で、現在のローカルのブランチを一覧で確認できます

Git使用の流れ③

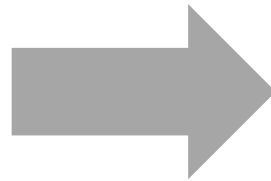
作業ブランチで作業します

※ソースの変更を
色で表現しています

リモート MASTER

ローカル

ローカル MASTER



commit

commit

commit

commit

作業ブランチ

作業がキリの良いときや、重要な変更を行う前後などでコミット(確定)します

```
git commit -m “コミットメッセージ”
```

後で好きなコミットしたタイミングにソースの状態を戻すこともできるため
重要な変更の前や、キリがいいとき、タスクが終わったときなど
こまめにコミットしておくの良いかもしれません（この辺は自由）

ただしコミットしていないとマージやプッシュといった操作ができないため、
それらの作業を行う前は必ずコミットします

コミットメッセージは作業の内容を簡単に書いておくと、後でたどりやすいです

Git使用の流れ④

merge



リモート MASTER

自分の作業中に他の人の作業が完了して、リモート **MASTER** が更新されることがあります

※ソースの変更を
色で表現しています

ローカル

ローカル MASTER

作業ブランチ

Git使用の流れ⑤

ローカルのMASTERを
リモートのMASTERと
同期させます

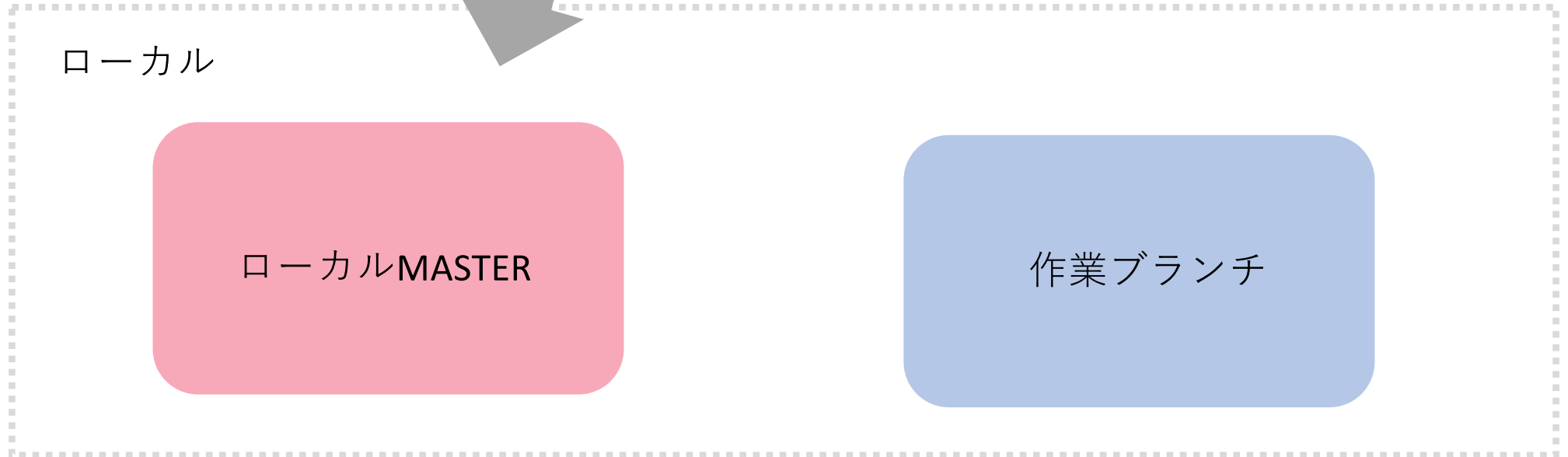
リモート MASTER

pull

ローカル

ローカルMASTER

作業ブランチ



リモートMASTERからローカルMASTERへプルします

```
git checkout master
```

まずはローカルのmasterブランチへ移動します

```
git pull
```

リモートからローカルへプルします（差分のダウンロード）

Git使用の流れ⑥

作業ブランチを
ローカルのMASTERと
合体させます

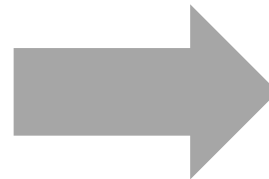
リモート MASTER

ローカル

ローカルMASTER

merge

作業ブランチ



リモートから取得してきた最新のmasterブランチと作業ブランチを合体させます

```
git checkout "作業ブランチ名"
```

まずは作業ブランチへ移動します

```
git merge master
```

masterをマージ（合体）します

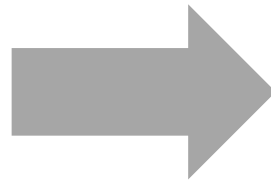
Git使用の流れ⑦

合体に成功すると、
自分と他人の修正が適用された
ソースになります

リモート MASTER

ローカル

ローカル MASTER



Completed !

作業ブランチ

Git使用の流れ⑧

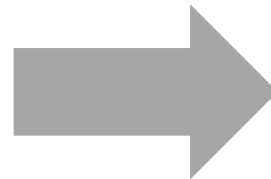
失敗した場合は、修正します

※自分と他人が同じファイルの
同じ場所を修正した場合など
修正が衝突することを
コンフリクトと呼びます

リモート MASTER

ローカル

ローカル MASTER



Conflict ~~~~~

作業ブランチ
※コンフリクト

コンフリクトが発生した場合は、エラーメッセージが表示されます

エラーで、どのファイルでコンフリクトが起きたかを教えてくれます

コンフリクトが発生したファイルを開くと、このような記述があります

```
<<<<<< HEAD
// 自分が行った修正
=====
// 他人が行った修正
>>>>>> 作業ブランチ名
```

修正内容を見てどちらの変更も残すか、片方だけにするかを判断します
(他人のコードで判断できない場合は相談します)

Git使用の流れ⑨

お互いの変更が反映された
ソースをリモートに送信します

リモートMASTER

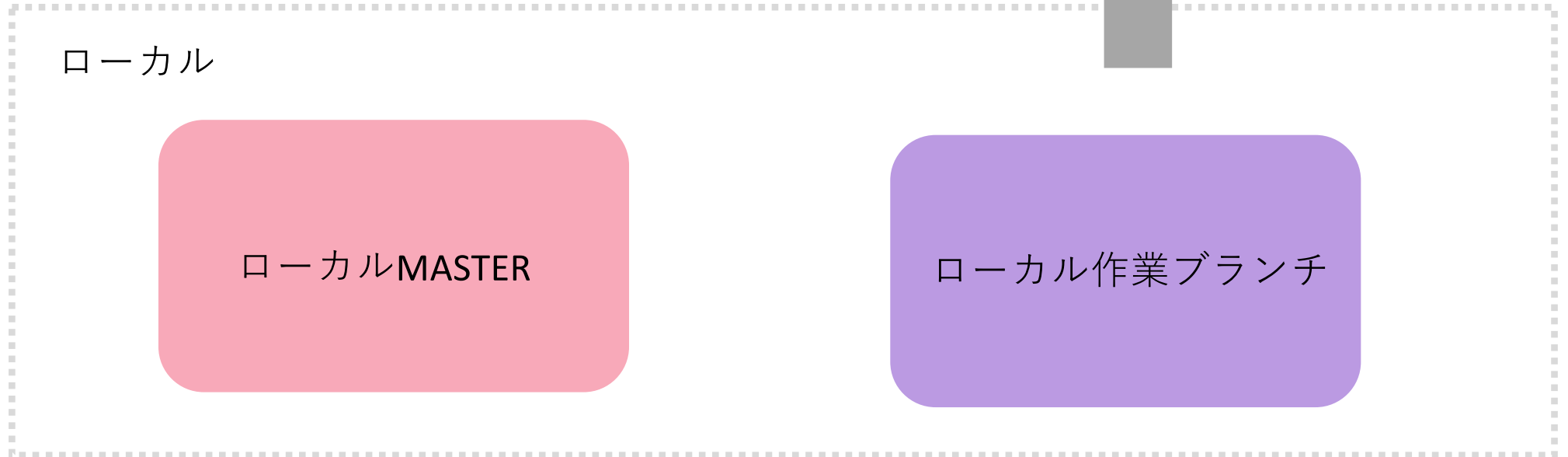
リモート作業ブランチ

push origin HEAD

ローカル

ローカルMASTER

ローカル作業ブランチ



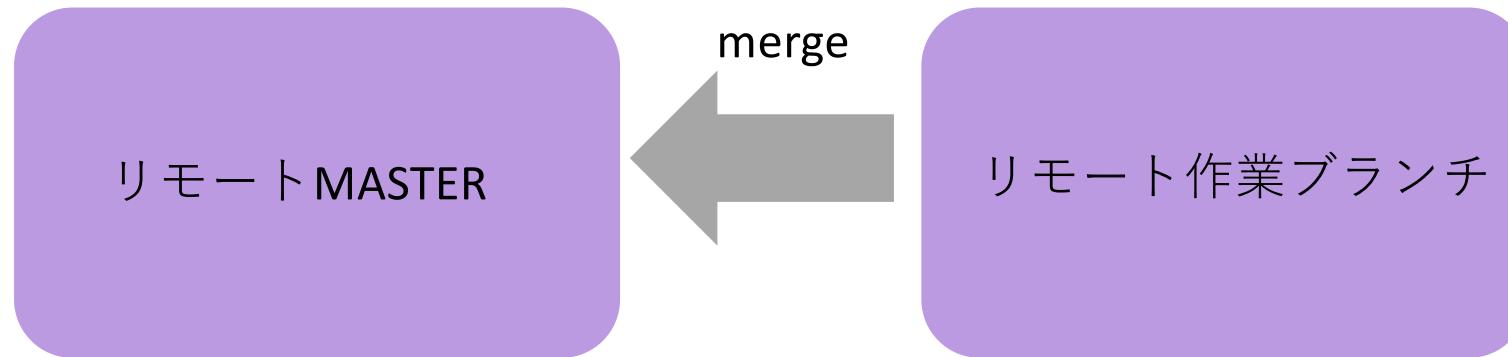
コンフリクトが解消し、マージが成功したらリモートへプッシュします

```
git push origin HEAD
```

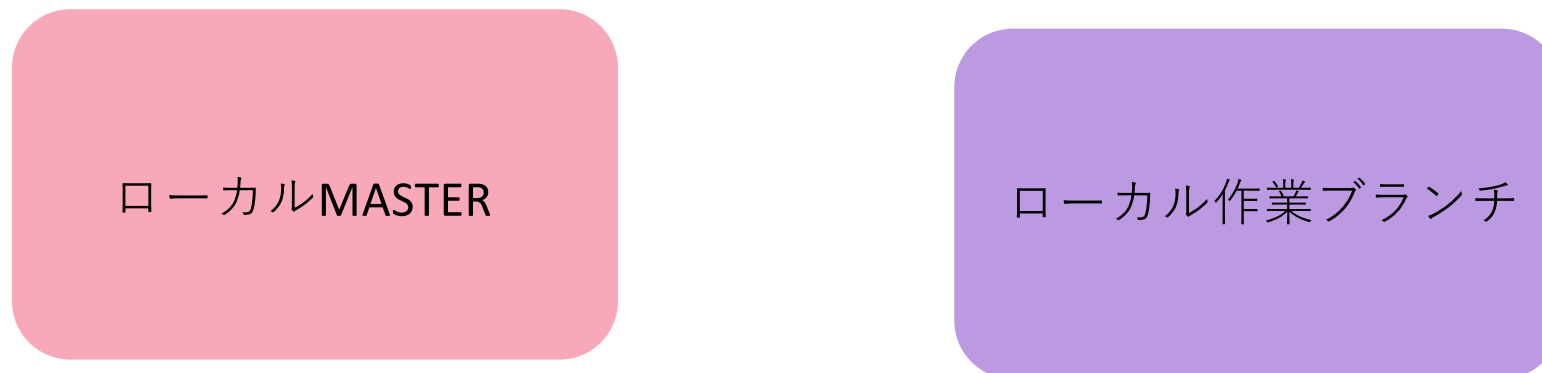
作業ブランチをリモートへプッシュします

Git使用の流れ⑩

リモートの**MASTER**ブランチへ
自分が行った修正をマージします



ローカル



GitLabを開きます

サイドメニューからマージリクエストを開き、New merge request を選択します



ソースブランチを選択して、自分が作業したブランチを選択します

prj-ekimae

プロジェクトの概要

リポジトリ

課題 9

マージリクエスト 0

CI / CD

運用

Packages & Registries

アクセス解析

Wiki

スニペット

メンバー

設定

ktt > ... > external > prj-ekimae > マージリクエスト > 新しい

New Merge Request

Source branch

ktt/wp/external/prj-ekimae

ソースブランチを選択

Target branch

ktt/wp/external/prj-ekimae

master

Merge branch 'add_ci' into 'master' ...

koyanagi-ktt が 4分前 にコミットしました

327499ef

Compare branches and continue

Target branchはmasterのままで、**Compare branches and continue** をクリックします

Submit マージリクエストをクリックします

P prj-ekimae

プロジェクトの概要

リポジトリ

課題 9

マージリクエスト 0

CI / CD

運用

Packages & Registries

アクセス解析

Wiki

スニペット

メンバー

設定

≪ サイドバーを隠す

New Merge Request

From master into 1-200718-marge [ブランチを変更](#)

タイトル

Master

[Start the title with Draft: or WIP:](#) to prevent a merge request that is a work in progress from being merged before it's ready.
追加 [description templates](#) コントリビュータのコミュニケーションを効率化する！

Description

Write プレビュー

B *I* ” </> [🔗](#) [☰](#) [☰](#) [☰](#) [☰](#) [☰](#) [☰](#) [☰](#) [☰](#)

変更の目的とレビューアが知っておくべきことを説明して下さい。

Markdown and [クイックアクション](#) are supported [📎 ファイルを添付](#)

Assignee

Unassigned

[Assign to me](#)

Milestone

Milestone

Labels

ラベル

マージオプション

☐ Squash commits when merge request is accepted. [?](#)

Submit マージリクエスト

Cancel

マージを選択します

The screenshot displays a GitHub pull request page for repository 'prj-ekimae'. The left sidebar contains navigation links: 'プロジェクトの概要', 'リポジトリ', '課題' (9 items), 'マージリクエスト' (1 item, highlighted), 'CI / CD', '運用', 'Packages & Registries', 'アクセス解析', 'Wiki', 'スニペット', 'メンバー', and '設定'. The main content area shows the pull request details: '未解決 (Open)' status, opened by 'Koga Yasunori', and the title 'Master'. It lists 0 summaries, 34 commits, 3 pipelines, and 148 changes. The merge step is highlighted with a red box, showing a 'マージ' (Merge) button and a checkbox for 'コミットを1つにまとめる' (Squash commits). Below this, a summary states '34個のコミットと1マージコミットを 1-200718-marge に追加する。' (Add 34 commits and 1 merge commit to 1-200718-marge). The right sidebar shows a 'To Do' list with items like '0人の担当者' (0 assignees), 'マイルストーン' (Milestones), 'タイムトラッキング' (Time tracking), 'ラベル' (Labels), 'マージリクエストをロック' (Lock pull request), and '1人の参加者' (1 participant).

これで、マージが成功すれば、
作業内容がリモートのmaster（テストサーバ上のソース）に反映されます

Git使用の流れ⑪

次の作業を行う場合は、
また②から同じ手順を
繰り返します

この流れで、同期を取りながら
チームで同時に作業ができます

