

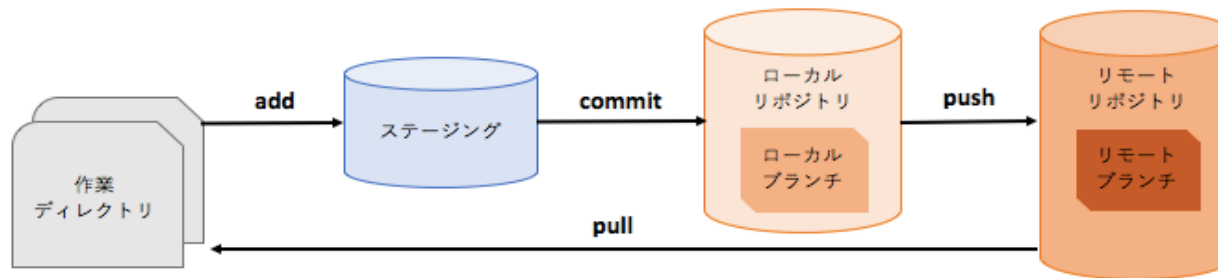
Git memo

since 2018/9/9

last update 2019/2/12

■ Git

- 元々 Linuxの開発で作られたバージョン管理システム
- SubVersionとは異なり、ローカルにも差分情報がある。(各自がサーバも兼ねてる感じ)



■ GitHub

<https://github.com/>

- Gitを使ったWebサービス。
 - 無料だが、リモートリポジトリは全て「公開」される。非公開にしたい場合は有料となる。
 - 2019/1/19 現在 制限付きだがPrivateリポジトリも無料で作れるようになった。
- WebAPI「GithubAPI」も公開している。

■環境構築

- 1.GitHubでアカウントを作成
- 2.Git for Windowsをインストール
 - これだけ入れればGitが使える（CUIでよければ...）

■ Gitクライアント

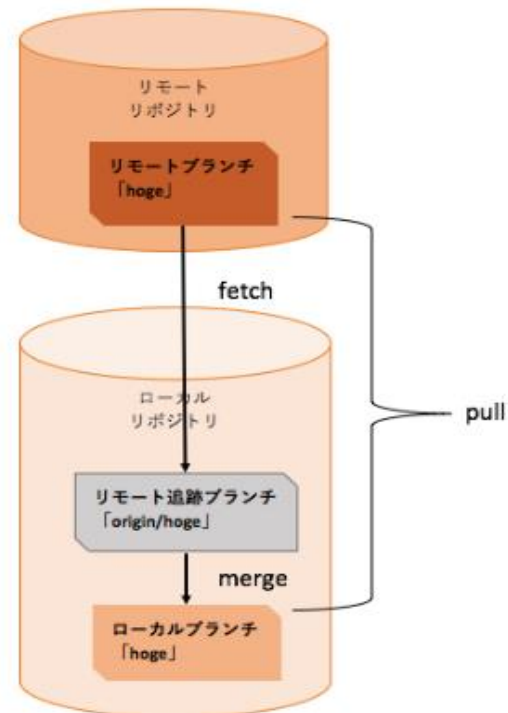
| | |
|----------------|---|
| GitHub DeskTop | GitHub自身が作っているGUIアプリ。 日本語化できれば言うことなし。 |
| SourceTree | GUIアプリ。高機能そう。 ここがわかりやすかった： https://naichilab.blogspot.com/2014/01/gitsourcetreegit.html?m=1 |
| TortoiseGit | エクスプローラに統合。 SubVersionのように使えて敷居が低そう。 オススメ 。 ここがわかりやすかった： https://backlog.com/ja/git-tutorial/ |
| Git Bash | CUIはメンドイ。(慣れるまでに挫折しそう...) |

■ 基本

| | |
|--------------------------------|---|
| リポジトリ repository | 履歴を入れるための入れ物(フォルダ単位) |
| リモートリポジトリ remote repository | サーバ上のリポジトリ。1人でGitを使う分には意識不要かも。(GitHubを使うなら必要) originという名前を付けるのが一般的 (ていうか自動的になる) リモートと省略される場合もある |
| ローカルリポジトリ local repository | ローカルにあるリポジトリ リポジトリ、ローカルと省略される場合もある コミットするとステージの情報がローカルリポジトリに確定される |
| ステージ stage | インデックスとも呼ばれる 作業ツリーとローカルリポジトリの間にある領域 作業ツリーのディレクトリ、ファイルはローカルリポジトリに入る前に一旦ステージに追加される |
| 作業ツリー working tree | ローカルの作業ディレクトリとその配下のサブディレクトリ、ファイルの総称。 ワークツリー、ワーキングツリーとも言う。 |
| クローン clone | リモートリポジトリを複製して、ローカルリポジトリを作成する(SubVersionでいうCheckOut) 変更履歴も複製される |

フェッチ
fetch

最新のリモートリポジトリを取得する。要はダウンロード。
プルと違って、取得するだけなので作業ツリーは変わらない。
(作業ツリーに反映する場合は ブランチをチェックアウトして マージ)
fetchすると追跡ブランチは更新されるが、ローカルのブランチ(のポインタ)は そのままとなる
リモートブランチから情報を取得してリモート追跡ブランチを更新するだけで、チェックアウトしているブランチにマージしません。
ワーキングツリーやインデックスには反映されません。

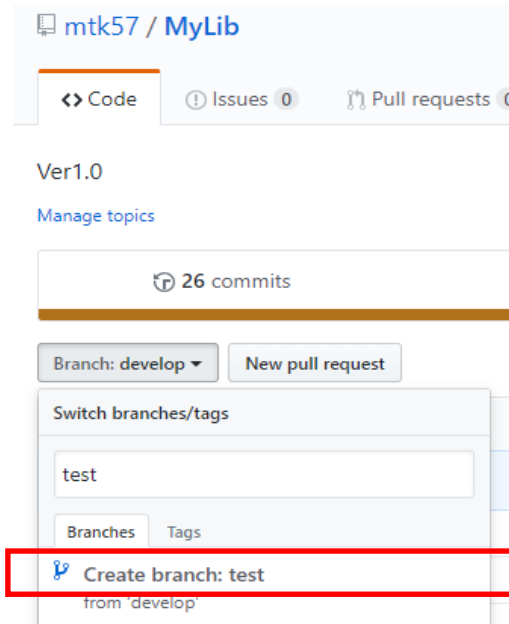


| | |
|--------------------------------------|---|
| コミット commit | <p>ローカルリポジトリにファイル/フォルダの変更を記録する（SubVersionには無い機能）</p> <p>コミットするときは必ずコミットログを書く必要があるが、直前のログの場合には簡単に修正することができます。(amend)</p> <p><u>TortoiseGitでのamend</u></p> <ol style="list-style-type: none"> 1.右クリックしてコミットウィンドウを開きます。 2.「最後のコミットのやり直し（日本語化していない場合はAmend Last Commit）」にチェックを入れる。 3.直前のコミットログがメッセージボックスに表示されるので、編集してコミットします。 |
| プッシュ push | <p>リモートリポジトリにローカルリポジトリの変更点を反映する（SubVersionでいうCommit）</p> <p>要はアップロード</p> |
| プル pull | <p>リモートリポジトリからローカルリポジトリに最新をコピーする（SubVersionでいうUpdate）</p> <p>要はダウンロード& マージ。フェッチとマージを同時にやってくれるのがミソ。</p> <p>→従って、pushの反対はfetchとなる（pullではない）</p> |
| プルリクエスト （プルリク、PR） pull request | <p>開発者のローカルリポジトリでの変更を他の開発者に通知する機能</p> <p>レビュー担当やマージ担当に依頼する場合に使う(つまり チーム開発用の機能)</p> <p>Gitの機能ではなくGitHubが最初に作った機能。</p> |
| チェックアウト checkout | <p>ブランチを切り替える（SubVersionのCheckOutとは全然違うので注意）</p> <p><u>TortoiseGitでのチェックアウト</u></p> <ol style="list-style-type: none"> 1.作業フォルダを右クリックして表示されるメニューで、「TortoiseGit」>「切り替え」 |

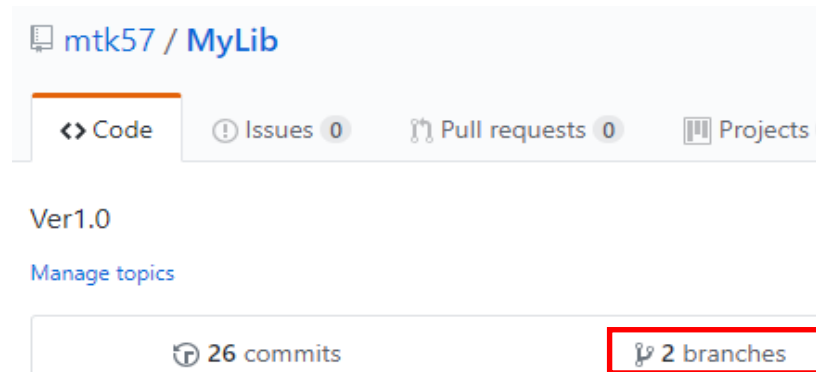
| | |
|------------------------|---|
| <p>ブランチ branch</p> | <p>ファイルを枝分かれさせて 複数バージョンを同時に管理すること 履歴の流れを分岐して記録していくためのもの 分岐したブランチは他のブランチの影響を受けないため、 同じリポジトリ中で複数の変更を同時に進めていくことができる ブランチの現在位置（ポインタ）を指す場合もブランチという</p> <p><u>ブランチの統合には、mergeを使う方法と、rebaseを使う方法の2種類があります。</u> どちらを使うかで統合後のブランチの履歴が大きく異なります。</p> <p><u>TortoiseGitでのブランチの削除</u></p> <ol style="list-style-type: none"> 1.作業フォルダを右クリックして表示されるメニューで、「TortoiseGit」>「リファレンスをブラウズ」をクリックすると、 リファレンスをブラウズ画面が表示されます。 2.リファレンスをブラウズ画面で、ブランチ名を右クリックして表示されるメニューで、「ブランチを削除」をクリックします。 →ローカル、リモートどちらのブランチも削除できる |
|------------------------|---|


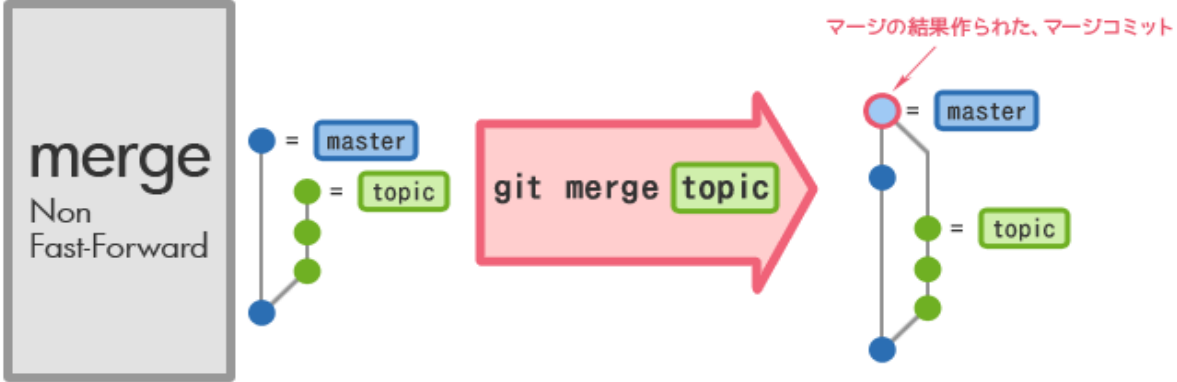
ブラウザでブランチ作成

switch branches/tagsのテキストボックスに、新規ブランチ名を入れると「Create branch」が表示される。



リネームは出来ないが、削除はできる。（ゴミ箱アイコン）



| | |
|----------------------|--|
| masterブランチ | <p>リポジトリに最初のコミットを行うと、Gitはmasterという名前のブランチを作成します。 そのため、以後のコミットはブランチを切り替えるまでmasterブランチに追加されていきます。</p> |
| <p>マージ merge</p> | <p>あるブランチに 他のブランチの変更を取り込むこと <u>複数の履歴の流れを合流</u>させることができます。 マージには2種類ある。</p> <p>Fast Forward</p>  <p>The diagram illustrates a Fast Forward merge. On the left, a box labeled 'merge Fast-Forward' is shown. To its right, a Git commit history is depicted: a blue dot labeled 'master' is at the bottom, with three green dots (representing commits) stacked vertically above it. A label 'topic' is next to the top green dot. A large pink arrow labeled 'git merge topic' points to the right. On the right side of the arrow, the commit history is shown again. The 'master' branch (blue dot) has moved up to the position of the top green dot, and the label 'master' is now next to it. The original 'topic' label remains next to the dot below it. A blue squiggly arrow points from the text '移動した だけ!' (Moved only!) to the new 'master' label.</p> <p>Non Fast Forward</p>  <p>The diagram illustrates a Non Fast Forward merge. On the left, a box labeled 'merge Non Fast-Forward' is shown. To its right, a Git commit history is depicted: a blue dot labeled 'master' is at the bottom, with three green dots stacked vertically above it. A label 'topic' is next to the top green dot. A large pink arrow labeled 'git merge topic' points to the right. On the right side of the arrow, the commit history is shown again. The 'master' branch (blue dot) has moved up to the position of the top green dot, and the label 'master' is next to it. The original 'topic' branch (green dots) remains below it. A red arrow points to the new 'master' commit with the text 'マージの結果作られた、マージコミット' (Merge commit created as a result of the merge).</p> <p>→ブランチはそのまま残る</p> |

TortoiseGitでのマージ操作

- 1.作業フォルダを右クリックして表示されるメニューで、「TortoiseGit」>「マージ」をクリックすると、マージ画面が表示されます。
- 2.マージ画面で、現在のHEADにどの版（コミット）をマージするのか「From」に入力します。
- 3.競合がなくマージを完了すると、「Gitコマンド実行中」画面に「成功」が表示されます。「閉じる」ボタンを押します。

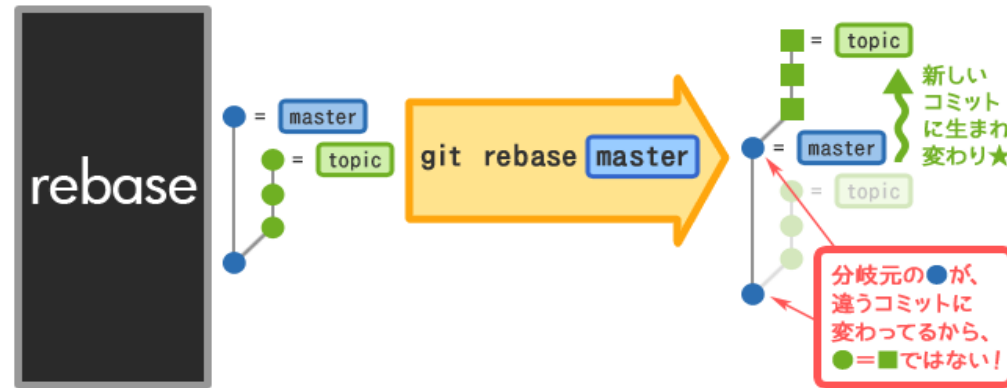
<競合がある場合の操作>

競合があると、Gitコマンド実行中画面が表示され、マージが中断されます。

- 1.競合の解決ボタンを押すと、コミット画面が表示されます。
- 2.コミット画面で競合があるファイルのファイル名をダブルクリックすると、TortoiseGitMerge画面が表示されます。(もしくはWinMerge)
左ブロックに「あちら側」としてマージ画面でFromに指定した版（コミット）の内容が表示され、
右ブロックに「こちら側」として現在のブランチのHEADの内容が表示されます。
下ブロックの「マージ済み」で競合の解消を行います。
- 3.「?????・・・」が表示されている部分を修正します。
行内をクリックすると行を選択することができ、選択した行は太枠で囲まれます。
コピーや貼り付けをしたり直接入力をして競合を解消します。
- 4.画面上にあるメニューの「解決済みとする」をクリックします。
- 5.コミット画面でコミットします。

リベース
rebase

ブランチの根元を別のブランチに付け替える



resetよりも**強力な過去改変!**

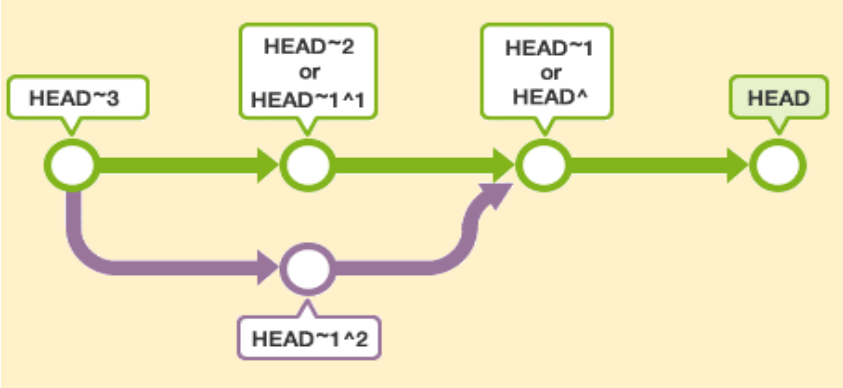
実はcherry-pickを連続して行っている

統合する目的でリベースを使うことができますが、共有リポジトリでリベースすることはできません。

その理由は、リベースすると失われる版（コミット）情報があり、リベースした後プッシュできなくなる恐れがあるためです。

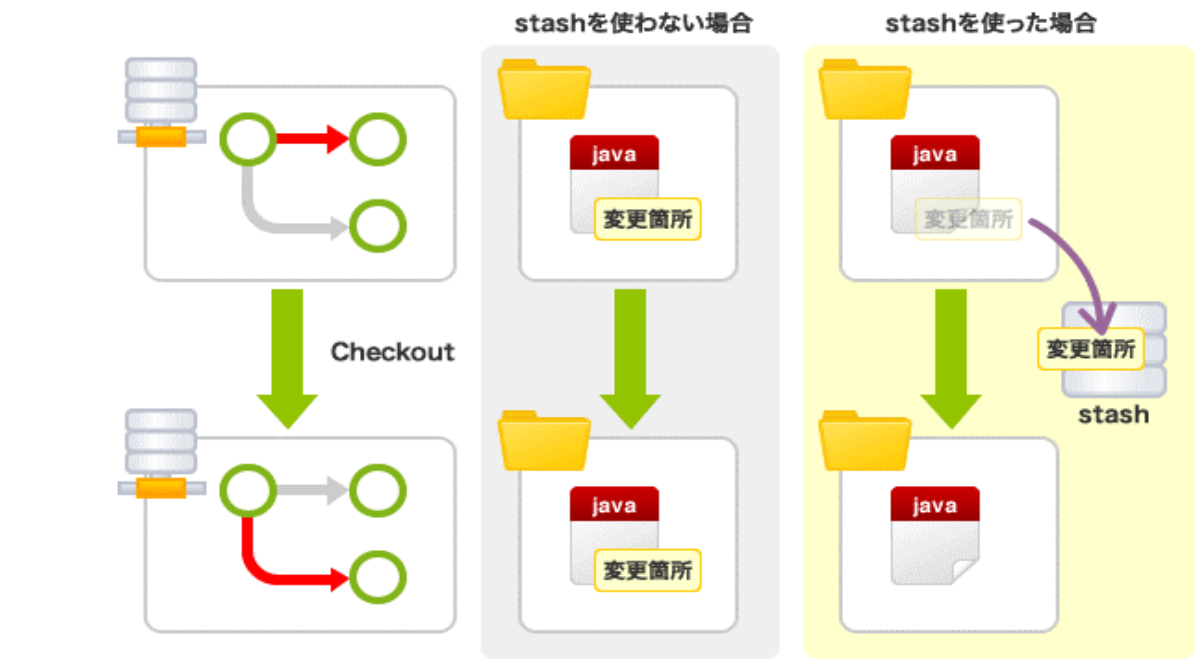
TortoiseGitでのマージ操作

1. 作業フォルダを右クリックして表示されるメニューで、「TortoiseGit」>「リベース（ブランチの付け替え）」をクリックするとリベース画面が表示されます。
2. リベース画面で、「ブランチ」に新しい主軸のブランチ、「上流」に前の主軸のブランチを入力します。
3. リベースにより反映される変更が一覧表示されます。
採用する変更を確認の上、「リベース開始」をクリックします。
4. 競合があるとリベースが中断されます。
リベース画面で競合があったファイルのファイル名をクリックすると、TortoiseGitMerge画面に競合内容が表示されます。
競合を解消してコミットします。リベースが完了するまで競合の解消を繰り返します。
5. リベースが完了したら、「終了」をクリックします。

| | | | | | | | |
|---------------|--|---------------|----------------------------------|-------------|--|--------|---------------------------|
| HEAD | <p>現在使用しているブランチの先頭を表す名前です。最後にコミットしたポイント最新のコミットに対するハッシュ値の別名です。</p> <p>HEADが移動することで、使用するブランチが変更されます。</p>  <p>The diagram illustrates the movement of the HEAD pointer in Git. It shows a sequence of four commit nodes connected by green arrows, representing a linear history. The nodes are labeled from left to right: HEAD~3, HEAD~2 or HEAD~1^1, HEAD~1 or HEAD^, and HEAD. A purple arrow originates from the HEAD~3 node, points down to a node labeled HEAD~1^2, and then points up to the HEAD~1 or HEAD^ node, indicating a merge or checkout operation that moves the HEAD pointer to a different branch.</p> | | | | | | |
| FETCH_HEAD | <p>リモートブランチの最新のコミットに対するハッシュ値の別名です。</p> <p>git cloneやgit fetch などで作成されます。</p> | | | | | | |
| ORIG_HEAD | <p>最新の一つ手前のコミットに対するハッシュ値の別名です。</p> | | | | | | |
| MERGE_HEAD | <p>ローカルブランチ<branch-name>の変更をmasterブランチにマージしたとします。</p> <p>このときのローカルブランチ<branch-name>の最新のコミットに対するハッシュ値の別名です。</p> | | | | | | |
| コミットの位置 | <table><tr><td>origin/master</td><td>リモートリポジトリの「master」ブランチの追跡ブランチの位置</td></tr><tr><td>origin/HEAD</td><td>リモートリポジトリのHEADの位置 通常「origin/master」と同じ位置を指す</td></tr><tr><td>master</td><td>ローカルリポジトリの「master」ブランチの位置</td></tr></table> | origin/master | リモートリポジトリの「master」ブランチの追跡ブランチの位置 | origin/HEAD | リモートリポジトリのHEADの位置 通常「origin/master」と同じ位置を指す | master | ローカルリポジトリの「master」ブランチの位置 |
| origin/master | リモートリポジトリの「master」ブランチの追跡ブランチの位置 | | | | | | |
| origin/HEAD | リモートリポジトリのHEADの位置 通常「origin/master」と同じ位置を指す | | | | | | |
| master | ローカルリポジトリの「master」ブランチの位置 | | | | | | |

| | |
|----------------------|--|
| サブモジュール submodule | <p>プロジェクトで管理しているGitリポジトリとは別に、独立したリポジトリをプロジェクトに含ませることができます。サブモジュールを含んだプロジェクトでは、プッシュ・プルをおこなってもサブモジュール側のリポジトリには影響がない、といった特徴があります。</p> <p>ブランチ単位で管理する通常のリポジトリと違い、サブモジュールはコミットID単位で管理するリポジトリ先の情報等は.gitmodulesというファイルに記載される。</p> <p>サブモジュールは親プロジェクトの特定のコミットとコミット単位で紐づく。</p> <p>サブモジュールを追跡する対象はブランチではなくコミットである。</p> |
| リビジョン revision | <p>コミットによって作られる、状態の単位を「リビジョン」という。</p> <p>より分かりやすい言葉だと「バージョン」に近い。</p> <p>ちなみに c77350968302fb61bccb3e604e2ecd297c9c3c02 みたいなのは「コミットのハッシュ値」とか「リビジョン番号」とか呼ぶ。</p> |
| スタッシュ stash | <p>現在の作業を一時的に退避する</p> <p>まだコミットしていない変更内容や新しく追加したファイルが、インデックスやワークツリーに残ったままで、他のブランチへのチェックアウトを行うと、その変更内容は元のブランチから、移動先のブランチに対して移動します。</p> <p>ただし、移動先のブランチで、同じファイルが既に何らかの変更が行われている場合はチェックアウトに失敗します。</p> <p>このような場合は、変更内容を一度コミットするか、またはstashを使って一時的に変更内容を退避させてからチェックアウトしなければなりません。</p> |

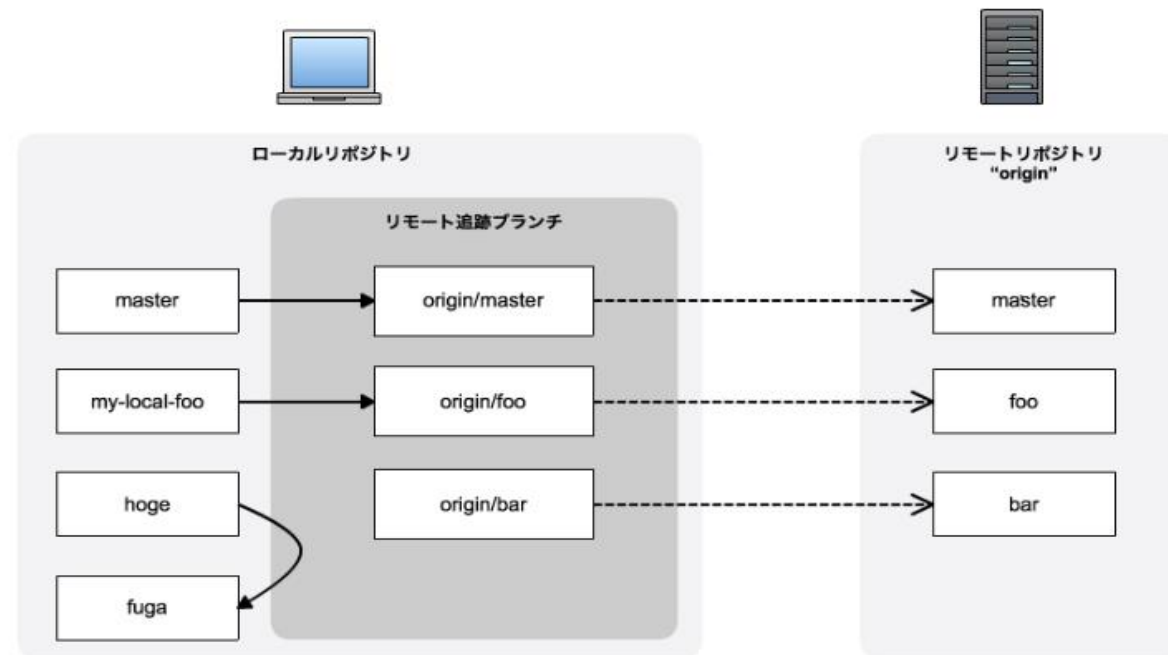
stashとは、ファイルの変更内容を一時的に記録しておく領域です。stashを使うことで、ワークツリーとインデックス中でまだコミットされていない変更を一時的に退避させることができます。退避させた変更は後から取り出して、元のブランチや別のブランチに反映させることができます。



タグ
tag

コミットを参照しやすくするために、わかりやすい名前を付けるものです。
軽量タグと、注釈付きタグの2種類のタグが使用できます。

| | |
|----------------|--|
| リバート revert | <p>既存のコミットを元に戻す。 「取り消したいコミットを打ち消すようなコミットを新しく作成する」という処理によって、コミットを元に戻します。 resetとは異なり、<u>打ち消しのコミットを行うので 履歴は残る。</u></p> <p><u>TortoiseGitでの操作</u> 1. ログを開き、リバートしたいログで右クリックして「この変更のコミットを戻す」 2. その後コミットすればリバート成功</p> |
| リセット reset | <p>HEADを履歴の中で移動させたり、ステージや作業ツリーの内容をHEADに合わせたりする revertとは異なり、単なる<u>ポインタの移動なのでコミット履歴は残らない。</u></p> <p><u>TortoiseGitでの操作</u> ※ローカルでの作業をいろいろしていたらよくわからなくなったので、元に戻したい場合。 1. TortoiseGit→Show log 2. origin/masterとなっている行で、「Reset “master” to this…→Hard」を選びます。 3. これで、自分のコミットをすべて無効にした状態になります。</p> |
| 追跡ブランチ | <p>リモートブランチの位置を記憶する特殊なローカルブランチ ローカルブランチmasterをリモートoriginにプッシュすると、リモートリポジトリにmasterブランチができる。 それと同時にローカルリポジトリのリモートのmasterと同じ場所に origin/masterという名前の追跡ブランチができる git fetchで更新されます。</p> |



A —————> B BはAの上流である (AはBを"追跡"する)

A - - - - -> B AはBをリモート追跡する

| | |
|------------------------|---|
| ポインタ | HEAD、ブランチ、タグは単なるポインタ。 |
| 切り離されたHEAD | HEADがどのブランチの先端とも一体でない状態。 任意のコミットをチェックアウトすると起こる。(チェックアウトするとコミットにHEADが移動するので) この状態でコミットしても ブランチとの関連は失われる。 |
| チェリーピック cherry-pick | <p>他のブランチの差分(パッチという)を取り込む。(コミットを抜き取る)</p> <div data-bbox="627 442 1480 885" data-label="Diagram"> <pre> graph LR subgraph master A((A)) --> B((B)) --> C((C)) --> D((D)) --> E((E)) end subgraph branch_1 C --> F((F)) --> G((G)) end D -.-> Dp((D')) Dp --- Note[Dのコミットのコピーが作られる] style Dp fill:#ffff00 </pre> </div> <p>cherry-pick</p> <p>別ブランチのコミットを取り込む branch_1で git cherry-pick D</p> <p><u>TortoiseGitでの操作</u> 取り込み先 : release 取り込み元 : develop とした場合 : 1.releaseブランチに切り替え 2.ログを表示 3.左上に現在のブランチ名 (release) が書いてあるリンクをクリックするとbrowse references (リファレンスをブラウズ) という画面が出てきます。 4.そこで取り込みたいコミットのあるブランチ (develop) を選択して、okをおします。 5.左上がdevelopになっているのを確認して、取り込みたいコミットを右クリックする。 6.Cherry Pick this commit... (「このコミットをチェリーピック(採用)」)と出てきますので押して、次の画面のcontinueボタンを押すと取り込むことができます。</p> |
| 融合 squash | マージ元ブランチでの変更を1コミットにまとめてマージする。 |

TortoiseGit



■ ログの見方

【凡例】


ログにおけるブランチ名やタグ名の色

| | |
|----------|---|
| 現在のブランチ |  |
| ローカルブランチ |  |
| リモートブランチ |  |
| タグ |  |



1. ローカルにCOMMIT

| グラフ | アクション | メッセージ |
|---|---|--|
|  |  | 作業ツリーの変更 master ローカルにコミット(リモートはmaster) |

2. リモートにPUSH

| グラフ | アクション | メッセージ |
|---|---|---|
|  |  | 作業ツリーの変更 master origin/master ローカルにコミット(リモートはmaster) |



3. masterからTopicを派生

| グラフ | アクション | メッセージ |
|---|---|--|
|  |  | 作業ツリーの変更 Topic master origin/master ローカルにコミット(リモートはmaster) |



4. TopicブランチをCOMMIT

| グラフ | アクション | メッセージ |
|---|---|---|
|  |  | 作業ツリーの変更 |
| | | Topic topicにコミット |
| | | master origin/master ローカルにコミット(リモートはmaster) |

5. TopicブランチをPUSH

| グラフ | アクション | メッセージ |
|---|---|---|
|  |  | 作業ツリーの変更 |
| | | Topic origin/Topic topicにコミット |
| | | master origin/master ローカルにコミット(リモートはmaster) |

6. masterからTopic_Subを派生

| グラフ | アクション | メッセージ |
|---|---|---|
|  |  | 作業ツリーの変更 |
| | | Topic Topic_Sub origin/Topic topicにコミット |
| | | master origin/master ローカルにコミット(リモートはmaster) |

7. Topic_SubブランチをCOMMIT

| グラフ | アクション | メッセージ |
|--|--|---|
|  |  | 作業ツリーの変更 |
| | | Topic_Sub topic_subにコミット |
| | | Topic origin/Topic topicにコミット |
| | | master origin/master ローカルにコミット(リモートはmaster) |

8. Topic_SubブランチをPUSH

| グラフ | アクション | メッセージ |
|---|---|--|
|  |  | 作業ツリーの変更 |
| | | Topic_Sub origin/Topic_Sub topic_subにコミット |
| | | Topic origin/Topic topicにコミット |
| | | master origin/master ローカルにコミット(リモートはmaster) |

9. Topicブランチに切り替え

| グラフ | アクション | メッセージ |
|-----|-------|---|
| | | 作業ツリーの変更 Topic origin/Topic topicにコミット master origin/master ローカルにコミット(リモートはmaster) |

10. TopicブランチをCOMMIT (2回目)

| グラフ | アクション | メッセージ |
|-----|-------|---|
| | | 作業ツリーの変更 Topic topicにコミット(2回目) origin/Topic topicにコミット master origin/master ローカルにコミット(リモートはmaster) |

11. TopicブランチをPUSH (2回目)

| グラフ | アクション | メッセージ |
|-----|-------|---|
| | | 作業ツリーの変更 Topic origin/Topic topicにコミット(2回目) topicにコミット master origin/master ローカルにコミット(リモートはmaster) |

12. TopicからDevelopを派生

| グラフ | アクション | メッセージ |
|-----|-------|--|
| | | 作業ツリーの変更 Develop Topic origin/Topic topicにコミット(2回目) topicにコミット master origin/master ローカルにコミット(リモートはmaster) |

13. masterに切り替え

| グラフ | アクション | メッセージ |
|-----|-------|---|
| | | 作業ツリーの変更 master origin/master ローカルにコミット(リモートはmaster) |

14. masterからHogeを派生

| グラフ | アクション | メッセージ |
|-----|-------|---|
| | | 作業ツリーの変更 Hoge master origin/master ローカルにコミット(リモートはmaster) |


15.HogeをCOMMIT

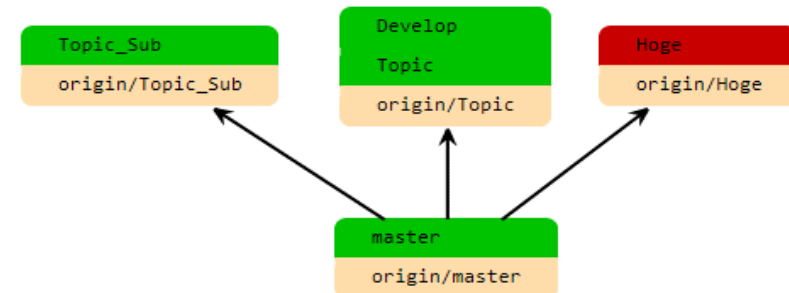
| グラフ | アクション | メッセージ |
|---|---|---|
|  |  | 作業ツリーの変更 Hoge Hogeにコミット master origin/master ローカルにコミット(リモートはmaster) |

16.HogeをPUSH



| グラフ | アクション | メッセージ |
|---|---|--|
|  |  | 作業ツリーの変更 Hoge origin/Hoge Hogeにコミット master origin/master ローカルにコミット(リモートはmaster) |

17.この時点でのリビジョングラフと全ブランチのログ



| グラフ | アクション | メッセージ |
|---|---|---|
|  |  | 作業ツリーの変更 Hoge origin/Hoge Hogeにコミット |
|  |  | Develop Topic origin/Topic topicにコミット(2回目) |
|  |  | Topic_Sub origin/Topic_Sub topic_subにコミット |
|  |  | topicにコミット |
|  |  | master origin/master ローカルにコミット(リモートはmaster) |



18.masterに切り替え、Hogeをmasterにマージ（Hogeは削除）

| グラフ | アクション | メッセージ |
|---|---|--|
|  |  | 作業ツリーの変更 master origin/Hoge Hogeにコミット origin/master ローカルにコミット(リモートはmaster) |

19.masterをPUSH

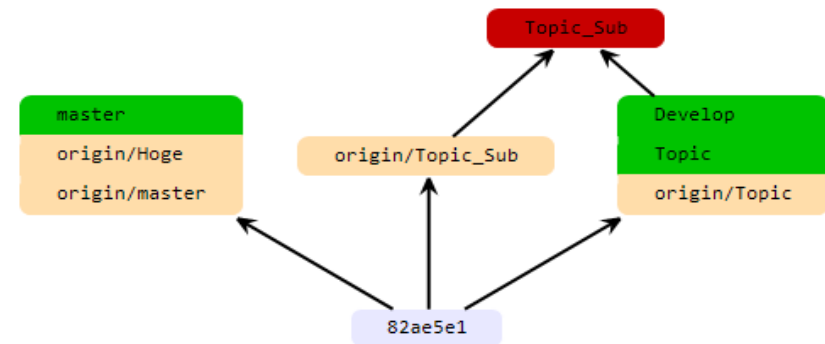
| グラフ | アクション | メッセージ |
|---|---|--|
|  |  | 作業ツリーの変更 master origin/master origin/Hoge Hogeにコミット ローカルにコミット(リモートはmaster) |

20. topic_subに切り替え

| グラフ | アクション | メッセージ |
|-----|-------|--|
| | | 作業ツリーの変更 Topic_Sub origin/Topic_Sub topic_subにコミット |
| | | topicにコミット ローカルにコミット(リモートはmaster) |

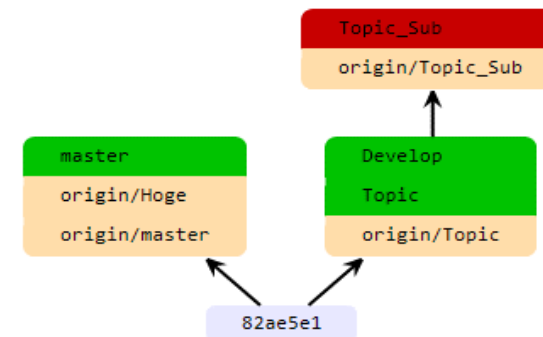
21. topicにMERGE

| グラフ | アクション | メッセージ |
|-----|-------|--|
| | | 作業ツリーの変更 Topic_Sub Merge branch 'Topic' into Topic_Sub |
| | | Develop Topic origin/Topic topicにコミット(2回目) |
| | | origin/Topic_Sub topic_subにコミット |
| | | topicにコミット ローカルにコミット(リモートはmaster) |



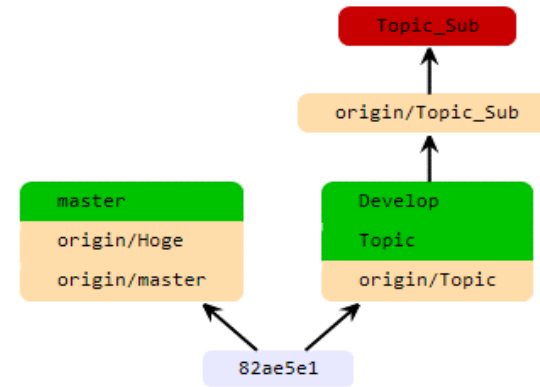
22. topic_subをPUSH

| グラフ | アクション | メッセージ |
|-----|-------|---|
| | | 作業ツリーの変更 Topic_Sub origin/Topic_Sub Merge branch 'Topic' into Topic_Sub |
| | | Develop Topic origin/Topic topicにコミット(2回目) |
| | | topic_subにコミット |
| | | topicにコミット |
| | | ローカルにコミット(リモートはmaster) |



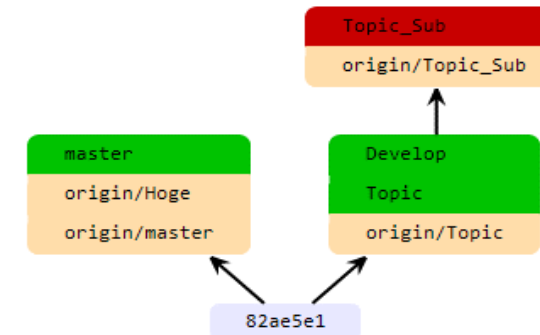
23. masterにマージしてCOMMIT

| グラフ | アクション | メッセージ |
|-----|-------|--|
| | | 作業ツリーの変更 |
| | | Topic_Sub hogeにマージ |
| | | origin/Topic_Sub Merge branch 'Topic' into Topic_Sub |
| | | Develop Topic origin/Topic topicにコミット(2回目) |
| | | topic_subにコミット topicにコミット ローカルにコミット(リモートはmaster) |



24. topic_subをPUSH

| グラフ | アクション | メッセージ |
|-----|-------|--|
| | | 作業ツリーの変更 |
| | | Topic_Sub origin/Topic_Sub hogeにマージ |
| | | Merge branch 'Topic' into Topic_Sub |
| | | Develop Topic origin/Topic topicにコミット(2回目) |
| | | topic_subにコミット topicにコミット ローカルにコミット(リモートはmaster) |



Command

ほとんどのコマンドは、ローカルリポジトリをカレントに変更してから実行する

add

| | |
|-----------------------|------------------------------|
| \$ git add [filename] | ファイルやディレクトリをインデックスに登録. |
| \$ git add -A | すべての変更を含むワークツリーの内容をインデックスに追加 |
| \$ git add -u | 以前コミットしたことがあるファイルだけインデックスに追加 |

branch

| | |
|---|-----------------------------------|
| \$ git branch | ローカルブランチを確認する。 カレントブランチには*がつく。 |
| \$ git branch &[new branch] | 現在のブランチの確認.&新しいブランチを作成する. |
| \$ git branch -a | ローカル、リモートの全てのブランチを確認する. |
| \$ git branch -r | リモートブランチを確認する. |
| \$ git branch -d [branch] | ブランチを削除する. |
| \$ git branch -m [branch] [new branchname] | ブランチの名前を変更する. |
| \$ git branch --set-upstream [my branch] [other branch] | 他のユーザーのブランチと自分のブランチを関連付ける. |

checkout

| | |
|---|------------------------------------|
| \$ git checkout -b [local branch] [remote branch] | ブランチを切り替える. |
| \$ git checkout [commit id] [filename] | コミットされた過去のファイルを復元する. |
| \$ git checkout [branch] | ブランチを切り替える. |
| \$ git checkout --ours [filename] | マージでコンフリクトしたときに情報を指定してファイル内容を採用する. |
| \$ git checkout --theirs [filename] | マージでコンフリクトしたときに下方を指定してファイル内容を採用する. |

cherry-pick

| | |
|--------------------------------|----------------------------|
| \$ git cherry-pick [commit id] | 別のブランチのコミットを現在のブランチにコピーする. |
|--------------------------------|----------------------------|

clone

| | |
|--|---------------|
| \$ git clone [repository PATH] [new repository PATH] | リポジトリをクローンする. |
|--|---------------|

commit

| | |
|---------------|--------------------------|
| \$ git commit | インデックスに追加されたファイルをコミットする. |
|---------------|--------------------------|

config

| | |
|---|---------------------|
| \$ git config -l | 使用されるリポジトリの設定を表示する. |
| \$ git config --global user.name [username] | ユーザ名の設定. |
| \$ git config --global user.email [email address] | メールアドレスの設定. |
| \$ git config --global color.ui auto | 出力結果を色づけする. |

fetch

| | |
|----------------------------------|---------------------------|
| \$ git fetch [remote repository] | リモートリポジトリの最新情報を追加する. |
| \$ git fetch --prune | リモートリポジトリの削除情報をローカルに更新する. |

init

| | |
|----------------------|------------------|
| \$ git init | ディレクトリにリポジトリを作成 |
| \$ git init --bare | ベアリポジトリの作成 |
| \$ git init --shared | グループ書き込み権限を有効にする |

merge

| | |
|-----------------------|------------------------|
| \$ git merge [branch] | 現在のブランチをほかのブランチとマージする. |
|-----------------------|------------------------|

mv

| | |
|-------------------------------------|----------------------------------|
| \$ git mv [filename 1] [filename 2] | ファイル名を変更(インデックスとワークツリーに同ファイル存在時) |
|-------------------------------------|----------------------------------|

pull

| | |
|---|--------------------|
| \$ git pull [remote repository PATH] [branch] | リモートリポジトリの変更を取り込む. |
|---|--------------------|

push

| | |
|--|----------------------------------|
| \$ git push [remote repository PATH] [branch] | リモートリポジトリに変更を書き込む. |
| \$ git push [remote repository] --tags | リモートリポジトリにすべてのタグをアップロードする. |
| \$ git push [remote repository] [tagname] | リモートリポジトリに指定したタグをアップロードする. |
| \$ git push [remote repository] :[branch or tagname] | 指定したブランチ,もしくはタグをリモートリポジトリから削除する. |

rebase

| | |
|------------------------------|---|
| \$ git rebase -i [commit id] | コミットIDから古い順にコミットが表示され、コミットの行を消すとコミットの取り消し、先頭のpickをほかのものに置き換えるとコミットメッセージなどの編集が可能になる。 |
| \$ git rebase --abort | 直前のgit rebaseの編集を中止する。 |
| \$ git rebase --continue | git rebaseの変更を適応する。 |

reflog

| | |
|------------------------|------------------------------------|
| \$ git reflog | 過去にHEADが指していたコミット一覧を表示する。 |
| \$ git reflog [branch] | ブランチを指定して過去にHEADが指していたコミット一覧を表示する。 |

remote

| | |
|---|----------------------------------|
| \$ git remote | リモトリポジトリの一覧表示 |
| \$ git remote update | リモトリポジトリ単位で情報を取得します。 |
| \$ git remote add [username] [remote repository PATH] | 名前とリモトリポジトリを関連付けする(リモトリポジトリの追加)。 |
| \$ git remote rename [remoterepository] [new name] | リモトリポジトリの名前を変更する。 |
| \$ git remote show [remote repository] | リモトリポジトリの情報を見る。 |
| \$ git remote prune [remote repository] | リモトリポジトリで排除されたブランチをローカルからも削除する。 |

reset

| | |
|-------------------------------------|-----------------------------|
| \$ git reset [commit id] | インデックスを現在のHEADの状態にする。 |
| \$ git reset HEAD [filename] | インデックスからファイルをアンステージする。 |
| \$ git reset --hard [commit id] | ワークツリーを含めたすべてをコミットIDの状態に戻す。 |
| \$ git reset --hard HEAD@{[number]} | git reflogで確認した番号の状態に戻す。 |
| \$ git reset --hard ORIG_HEAD | 直前の状態に戻す。 |

revert

| | |
|---------------------------|-------------------|
| \$ git revert [commit id] | コミットIDのコミットを取り消す。 |
|---------------------------|-------------------|

rm

| | |
|------------------------------|-------------------------|
| \$ git rm [filename] | ワークツリーとインデックスからファイルを削除。 |
| \$ git rm --cache [filename] | インデックスのファイルを削除。 |

show

| | |
|-----------------------|-------------------|
| \$ git show | 最新のコミット内容を表示. |
| \$ git show [tagname] | タグを指定してコミット内容を表示. |

stash

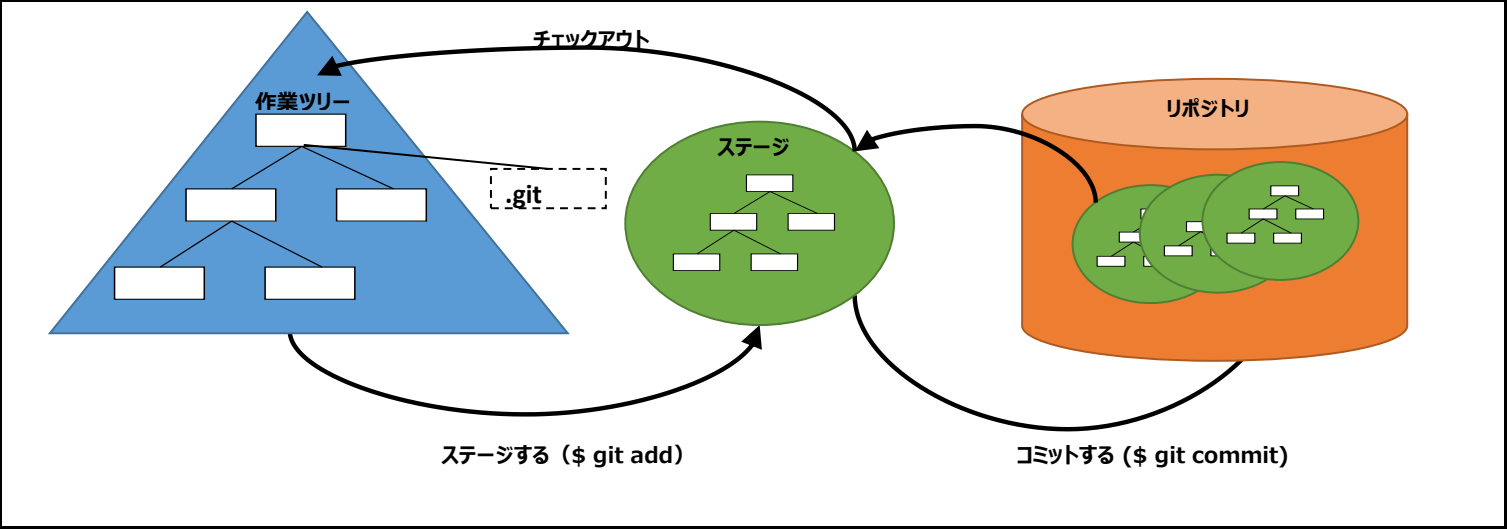
| | |
|-----------------------------------|---------------------------------|
| \$ git stash | 現在の状態を保存する. |
| \$ git stash save "[message]" | メッセージ付きで現在の状態を保存する. |
| \$ git stash list | 保存した状態の一覧を表示する. |
| \$ git stash pop | 最新の保存状態を復元する. |
| \$ git stash pop stash@[numbar] | 番号を指定して保存状態を復元する. |
| \$ git stash apply | 保存状態をリストに残したまま最新の保存状態を復元する. |
| \$ git stash apply stash@[number] | 保存状態をリストに残したまま指定した番号の保存状態を復元する. |
| \$ git stash drop stash@[number] | 保存状態を削除する. |
| \$ git stash clear | 保存状態をすべて削除する. |

tag

| | |
|----------------------------------|--|
| \$ git tag | タグの一覧を表示. |
| \$ git tag -n[number]@ | タグとそのメッセージ[行数指定]の一覧を表示する(行数指定なしの場合 1 行). |
| \$ git tag -l [filter] | タグを,フィルターをかけて表示する. |
| \$ git tag [tagname] | 現在のコミットIDにタグを関連付けする. |
| \$ git tag [tagname] [commit id] | コミットIDを指定してタグを関連付けする. |
| \$ git tag -a [tagname] | 現在のコミットIDにメッセージ付きのタグを関連付けする. |
| \$ git tag -d [tagname] | 指定したタグを削除する. |

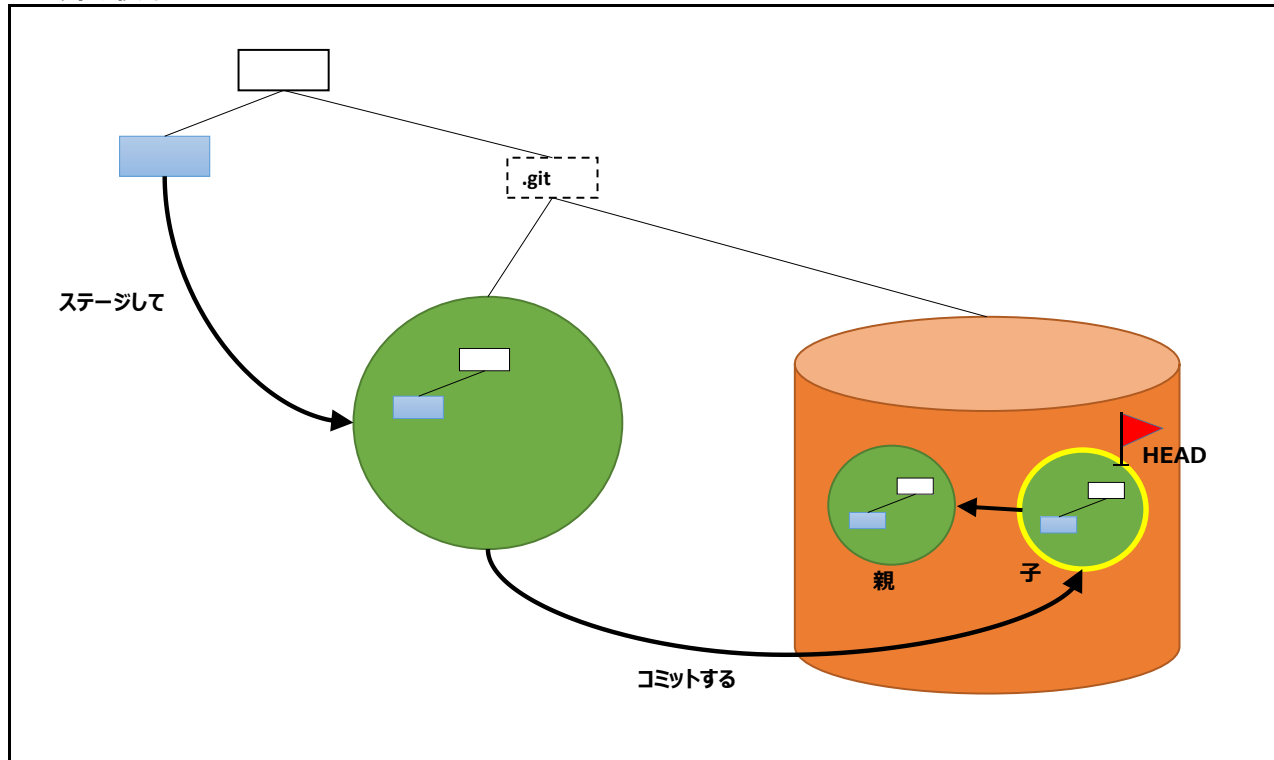
Resource

■プロジェクトのディレクトリ構成



\$ git init を作業ツリーの最上位の階層で実行すると、「.git」隠しフォルダが作成される。
.gitフォルダ配下には、でステージとリポジトリを管理するためのファイルが作成される。

■ ヘッドの移動



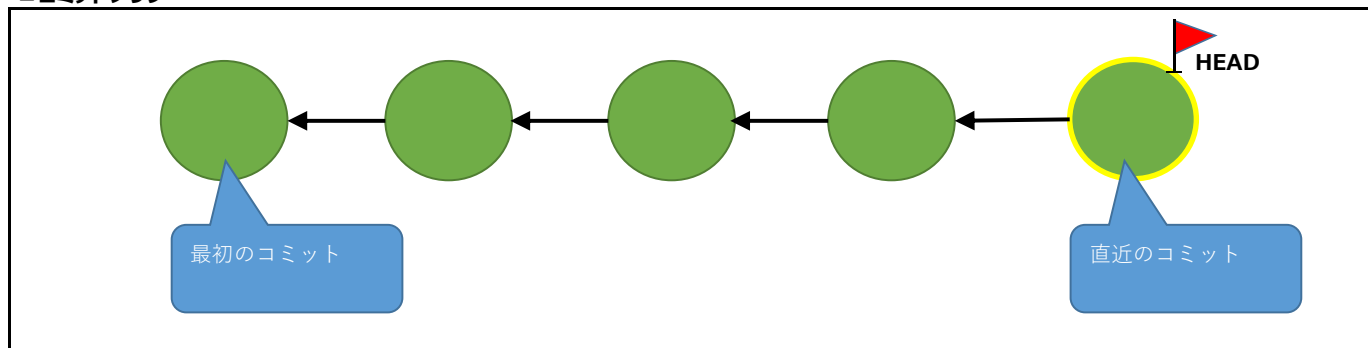
Gitは「現在のコミット」（最新のコミット）を指すポインタ（上図の旗）を保持していて、このポインタを「HEAD」という。

コミットすると、HEADはそのコミットを指す。

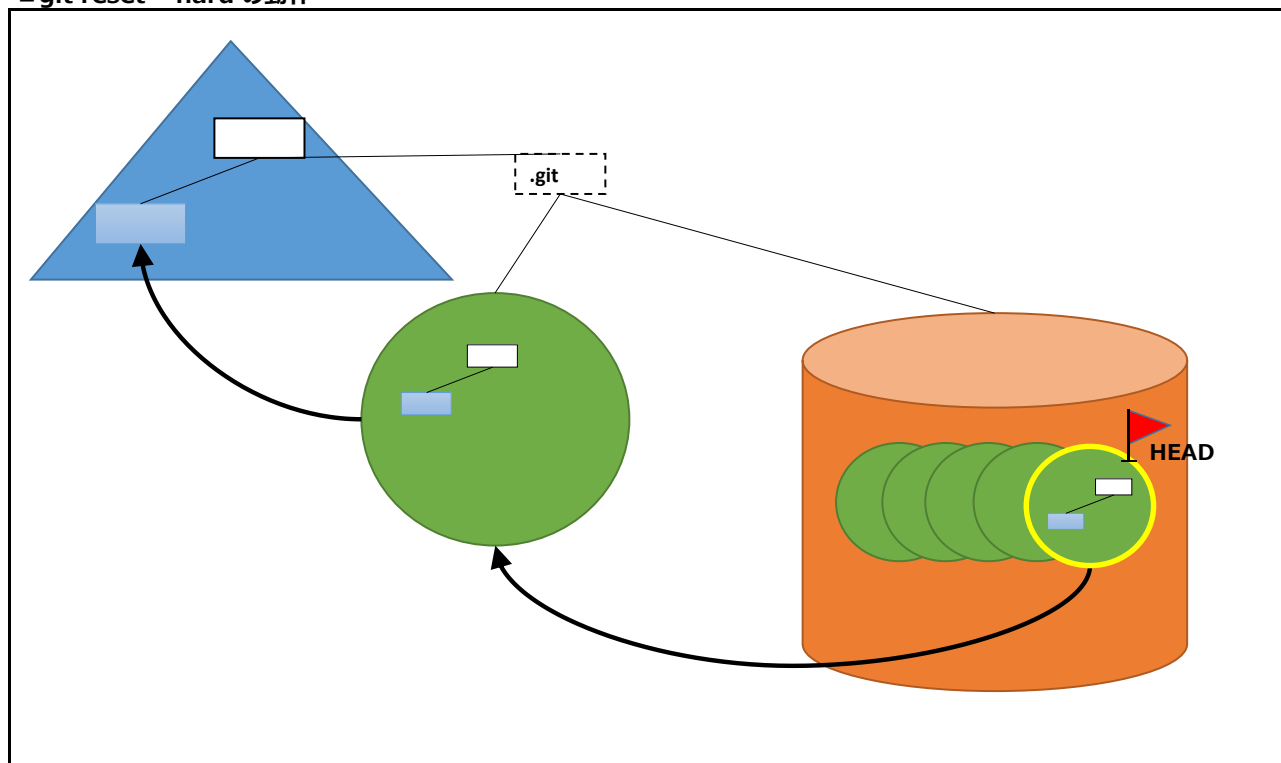
元々HEADが指していたコミットを新しいコミットの親と呼び、新しいコミットをそのコミットの子と呼ぶ。

子コミットは親コミットのポインタを持つ。（上図の矢印）

■コミットグラフ

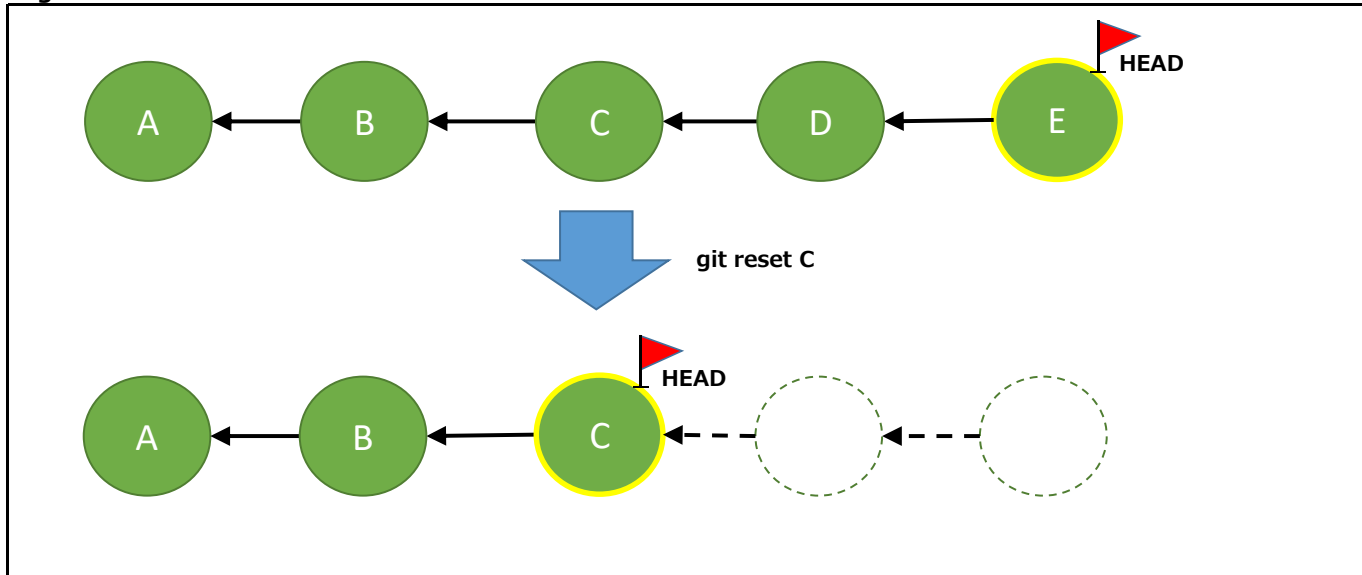


■git reset --hard の動作



git reset --hardを行うと、作業ツリー、ステージがHEADの状態となる。(作業ツリーやステージに変更しているファイルがあると悲惨)

■ git reset C の動作

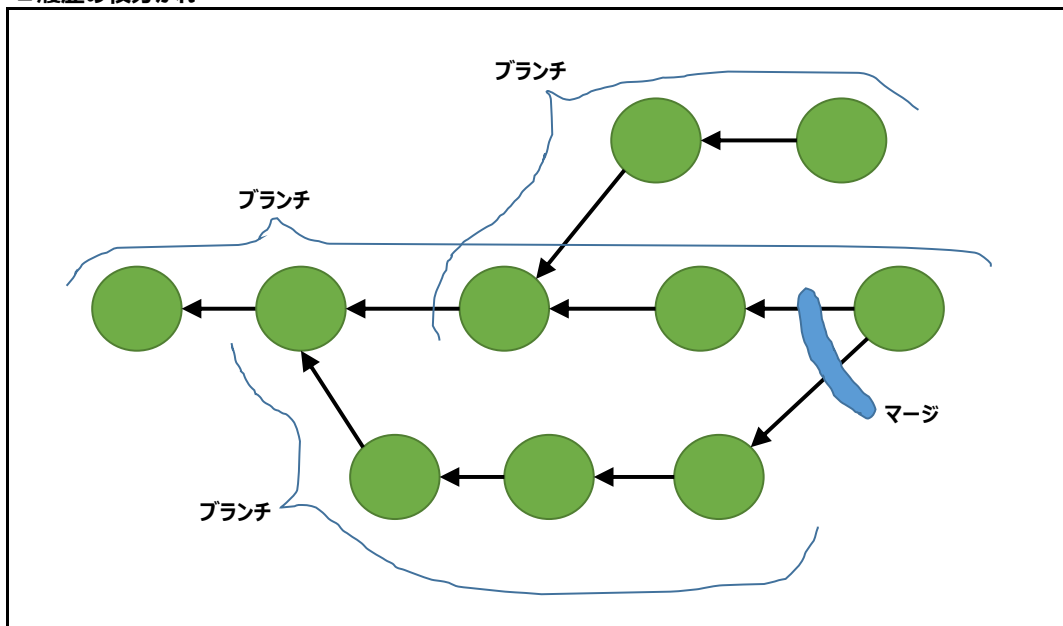


resetコマンドは、HEADの移動を行える。

オプション指定なしなので、作業ツリーは変えない。（--softをつけるとステージも作業ツリーも変えない）

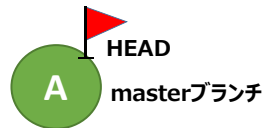
誤って、git resetした後で、HEADを元のコミットに戻す手段もある。

■履歴の枝分かれ

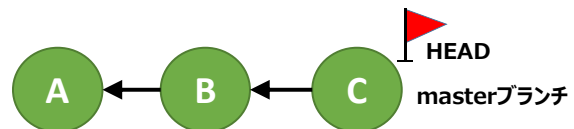


■ ブランチの成長・枝分かれ・切り替え

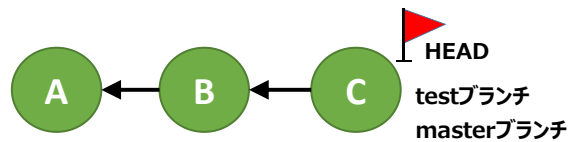
① 最初のコミット



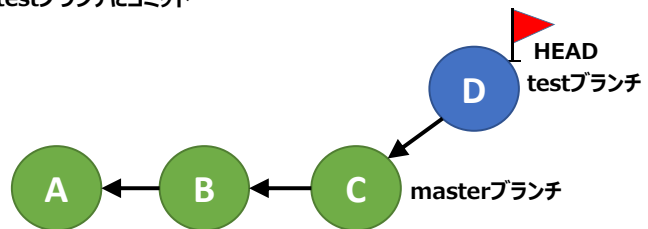
② コミットを2回行う (masterブランチが成長していく)



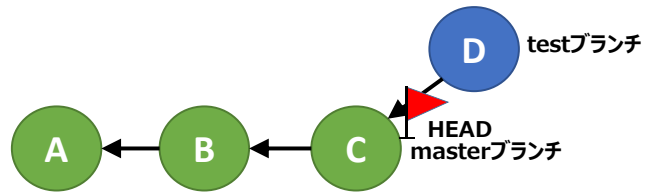
③ testブランチを作成し、HEADを切り替え (下図では分かりづらいが)



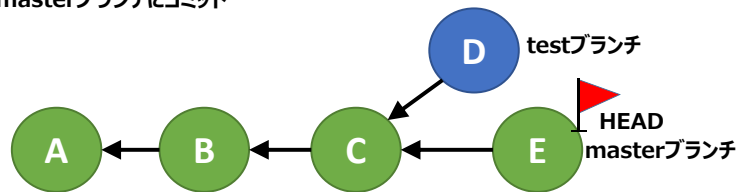
④ testブランチにコミット



⑤ masterブランチに切り替え

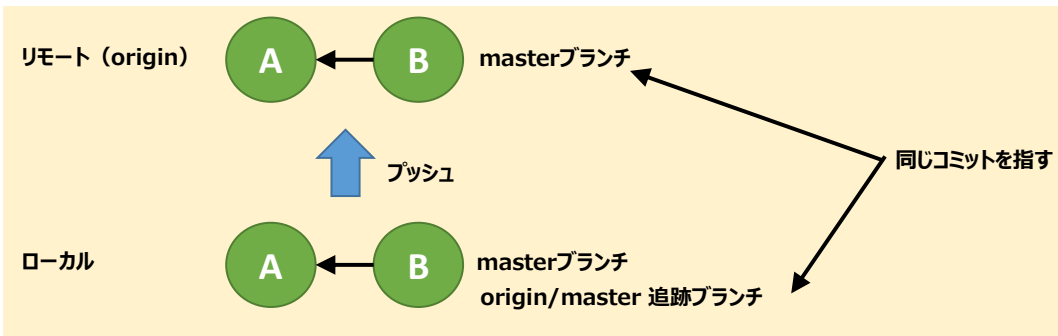


⑥ masterブランチにコミット

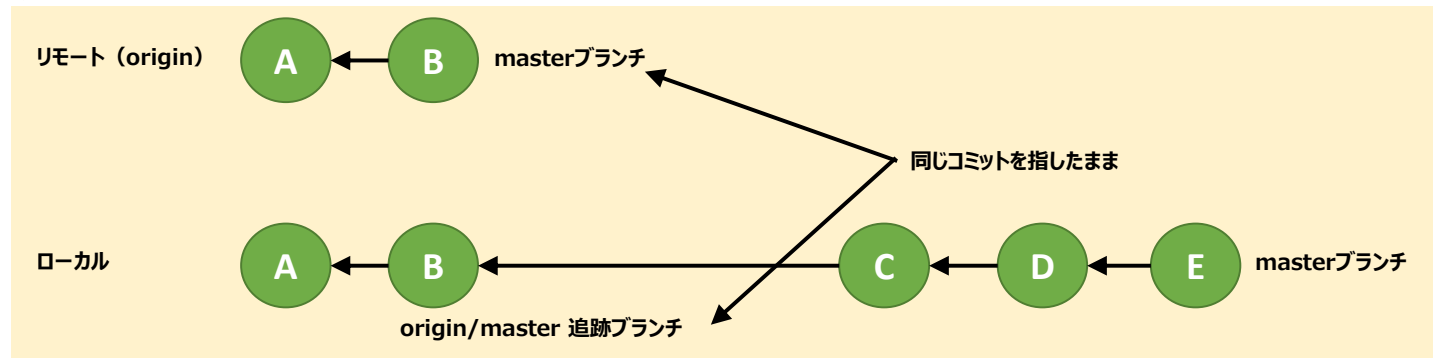


■ 追跡ブランチ

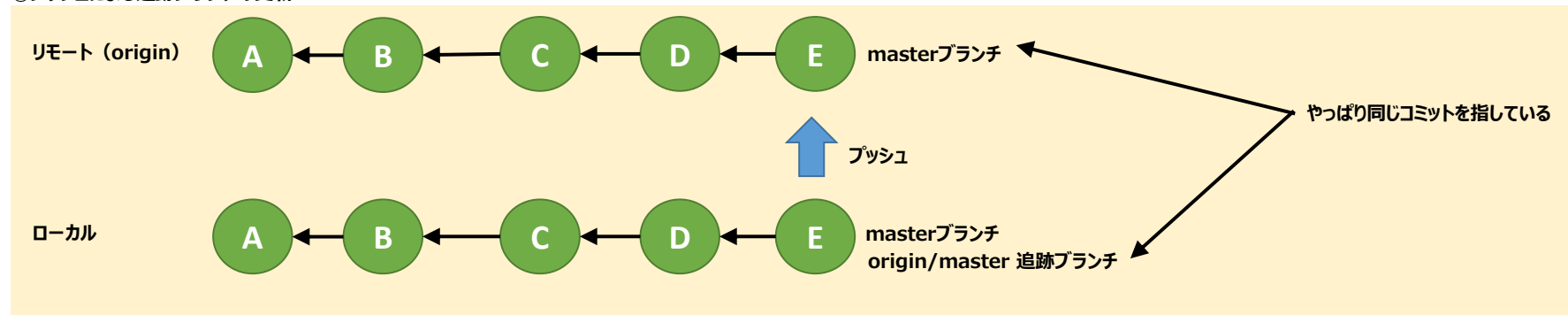
① 追跡ブランチの作成



② ローカルでの開発が進んでも



③ プッシュによる追跡ブランチの更新



[illegible]

ローカルのmasterの位置は変わらない

[illegible]

リモート (origin) A ← B ← C ← D masterブランチ

ローカル

```
graph RL; B((B)) --> A((A)); C((C)) --> B; D((D)) --> C;
```

origin/master 追跡ブランチ

masterブランチ

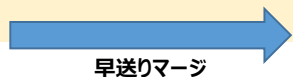
リモート (origin) A ← B ← C ← D masterブランチ

ローカル



```
graph RL; B((B)) --> A((A)); C((C)) --> B; D((D)) --> C;
```

origin/master 追跡
master ブランチ



Pull Requestを使った開発の流れ

| | | |
|---|-------------|--|
| ① | 開発者 | 作業対象のソースを clone または pull します。 |
| ② | | 作業用のブランチを作成します。 |
| ③ | | 機能追加、改修といった開発作業を行います。 |
| ④ | | 作業が完了したら push します。 |
| ⑤ | | プルリクエストを作成します。 |
| ⑥ | レビュー・マージ担当者 | 通知されたプルリクエストから変更を確認しレビューします。 |
| ⑦ | | レビュー結果を判断し、必要ならば開発者にフィードバックします。 |
| ⑧ | | レビューの結果、問題がない場合はマージします。 |
| ⑨ | | レビューの結果、対応自体が不要となるなど、プルリクエスト自体が必要ない場合はクローズします。 |

上記の ③ ～ ⑦ の工程を、必要な分だけ繰り返します

BookMarklet

・ブラウザのブックマークに登録して使う簡単なコード。（GitHubの機能ではない）

| # | 機能 | コード | 備考 |
|----|----------------------------|---|-------------------------|
| 1 | 隠れているoutdated diffをすべて表示する | javascript:\$(\$('.discussion-item-body').each(function(){\$(this).css('display','block')})); | 2019/1/19 現在 上手く動かない... |
| 2 | 日付を絶対表示にする | javascript:\$('time').each(function(){\$(this).html(\$(this).attr('title'))}) | 2019/1/19 現在 上手く動かない... |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |