

Git memo

since 2018/9/9

last update 2019/1/20

■ 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 制限があるがPrivateリポジトリも無料で作れるようになった。

■ 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/git-sourcetree-git.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) 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。
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
コミット commit	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)
プッシュ push	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) 要はアップロード

プル pull	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） 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。 →従って、pushの反対はfetchとなる（pullではない）
プルリクエスト （プルリク、PR） pull request	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 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） Gitの機能ではなくGitHubが最初に作った機能。
チェックアウト checkout	ブランチを切り替える（SubVersionのCheckOutとは全然違うので注意）
ブランチ branch	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
masterブランチ	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。

マージ
merge

あるブランチに 他のブランチの変更を取り込むこと

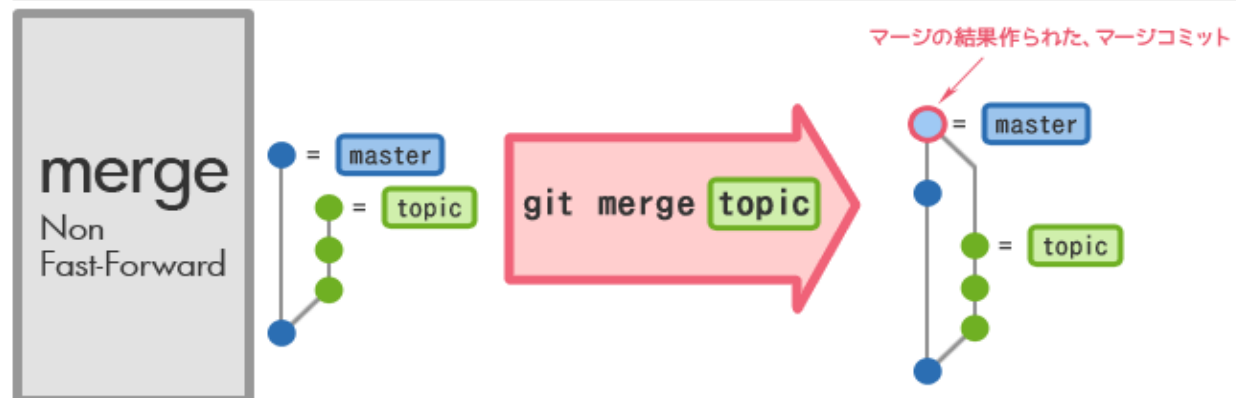
合流

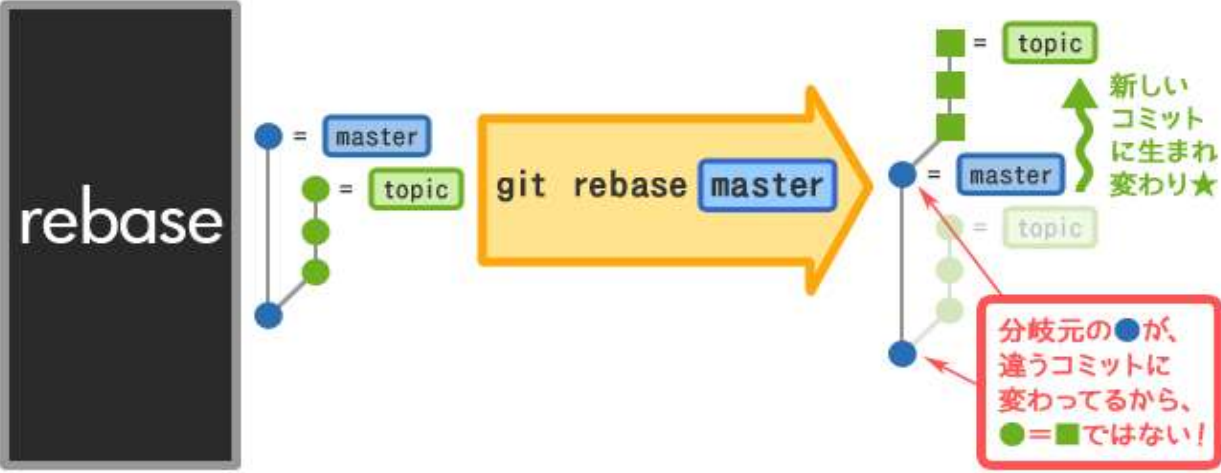
マージには2種類ある。

Fast Forward



Non Fast Forward



リベース rebase	<p>ブランチの根元を別のブランチに付け替える</p>  <p>rebase 実は cherry-pick とは別物</p>						
HEAD	<p>現在使用しているブランチの先頭を表す名前です。 デフォルトではmasterの先頭を表しています。 HEADが移動することで、使用するブランチが変更されます。 どのブランチの「最新でもないコミット」をチェックアウトすると付く(仮のブランチを表す) →変更できるが 仮のブランチなのでコミットしないこと！ →ブランチに変えることもできる</p>						
コミットの位置	<table border="1"> <tbody> <tr> <td>origin/master</td><td>「origin」リモートリポジトリの「master」ブランチの位置</td></tr> <tr> <td>origin/HEAD</td><td>「origin」リモートリポジトリをクローンした時にダウンロードされるコミットの位置 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> </tbody> </table>	origin/master	「origin」リモートリポジトリの「master」ブランチの位置	origin/HEAD	「origin」リモートリポジトリをクローンした時にダウンロードされるコミットの位置 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	「origin」リモートリポジトリの「master」ブランチの位置						
origin/HEAD	「origin」リモートリポジトリをクローンした時にダウンロードされるコミットの位置 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	現在の作業を一時的に退避する
タグ tag	<p>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。</p> <p>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。</p>
リバート revert	<p>既存のコミットを元に戻す。</p> <p>「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。</p> <p>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>
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</p> <p>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>
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</p> <p>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。</p> <p>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</p>
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</p> <div data-bbox="654 384 1550 499"><h1>cherry-pick</h1></div> <p>別ブランチのコミットを取り込む branch_1で git cherry-pick D</p>  <pre>graph LR subgraph master A((A)) --> B((B)) B --> C((C)) C --> D((D)) D --> E((E)) end subgraph branch_1 C --> F((F)) F --> G((G)) G --> Dp((D')) end style Dp fill:#ffff00 label Dp "Dのコミットのコピー が作られる"</pre>

Command

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

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

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.
\$ 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]	ブランチを変更する.
\$ 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.

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.

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.

rm	\$ git reset --hard ORIG_HEAD	直前の状態に戻す.
	\$ 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.
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)
revert	\$ git revert [commit id]	コミットIDのコミットを取り消す.
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.
clone	\$ git clone [repository PATH] [new repository PATH]	リポジトリをコピーする.
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.
pull	\$ git pull [remote repository PATH] [branch]	リモートリポジトリの変更を取り込む.
remote	\$ git remote	リモートリポジトリの一覧表示

\$ git remote add [username] [remote repository PATH]	名前とリモートリポジトリを関連付けする(リモートリポジトリの追加).
\$ git remote rename [remoterepository] [new name]	リモートリポジトリの名前を変更する.
\$ git remote show [remote repository]	リモートリポジトリの情報を見る.
\$ git remote prune [remote repository]	リモー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.

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.

branch

\$ 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	すべ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.

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.
-----------------------	------------------------

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.

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.

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.

cherry-pick

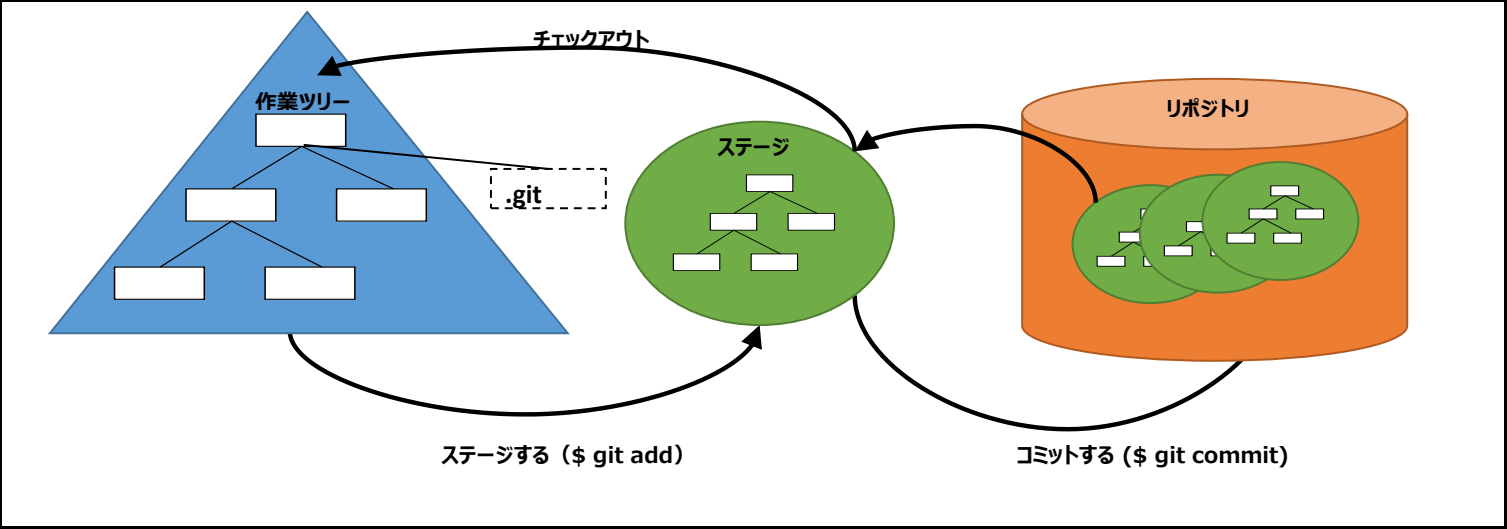
\$ git cherry-pick [commit id]	別のブランチのコミットを現在のブランチにコピーする.
--------------------------------	----------------------------

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.

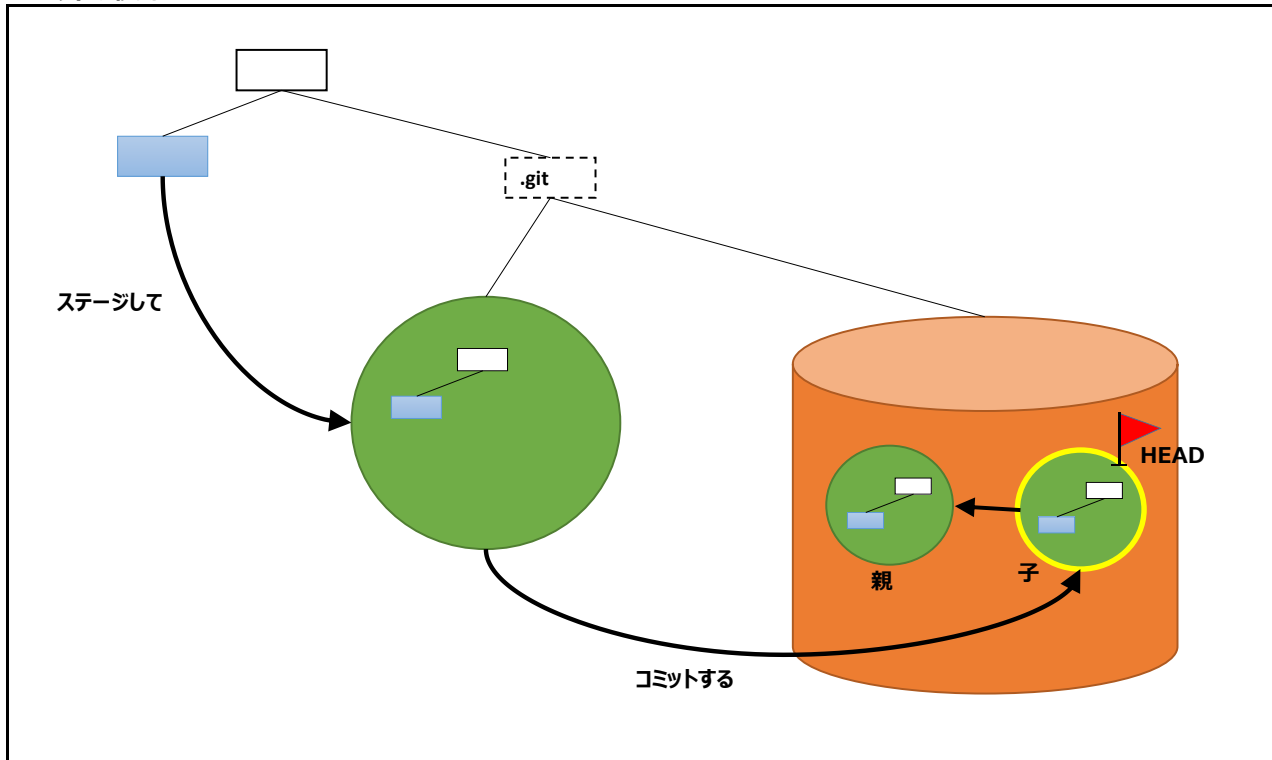
Resource

■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



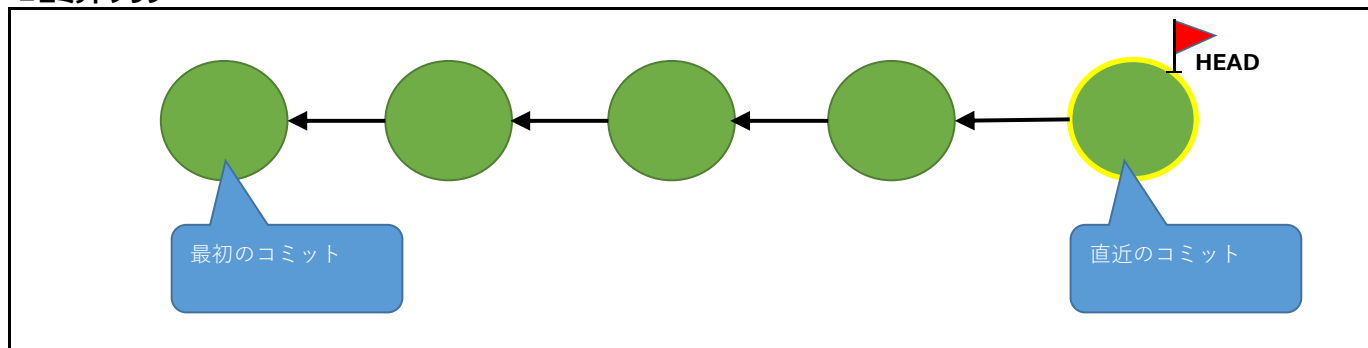
Gitは「現在のコミット」（最新のコミット）を指すポインタ（上図の旗）を保持していて、このポインタを「HEAD」という。

コミットすると、HEADはそのコミットを指す。

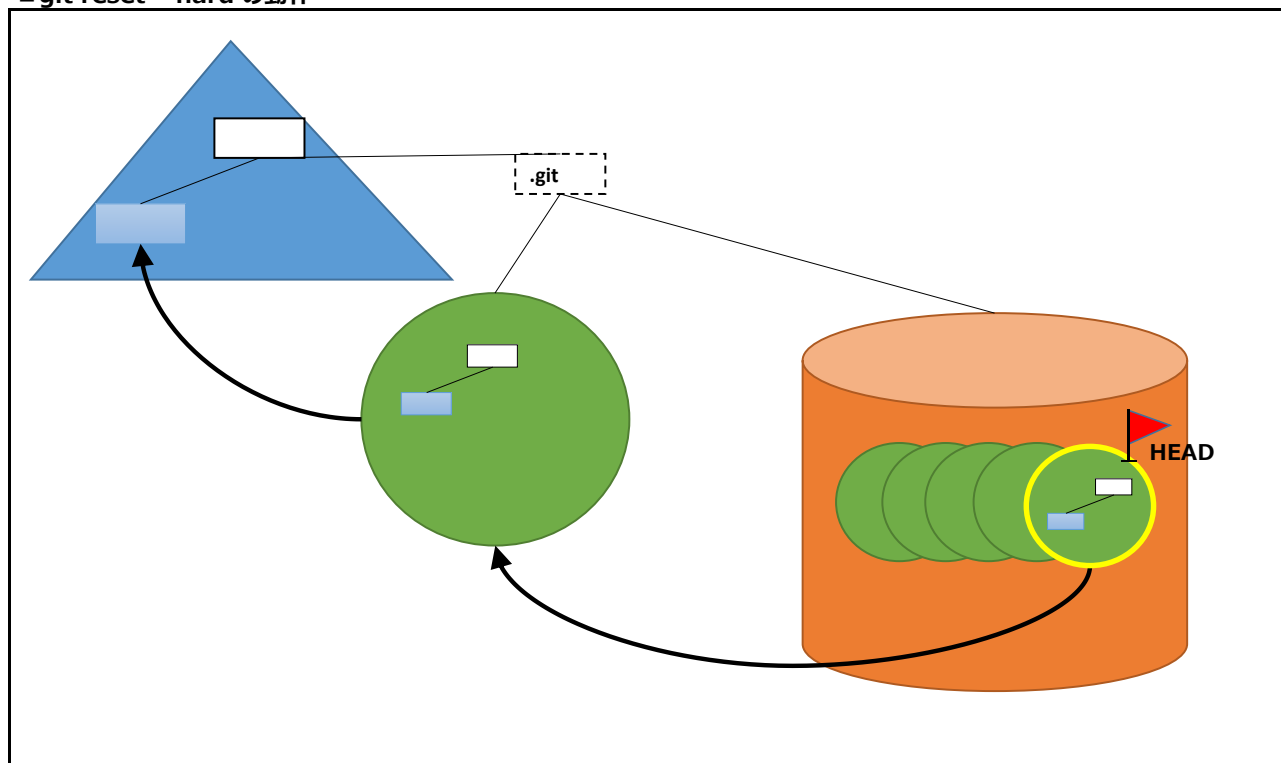
元々HEADが指していたコミットを新しいコミットの親と呼び、新しいコミットをそのコミットの子と呼ぶ。

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

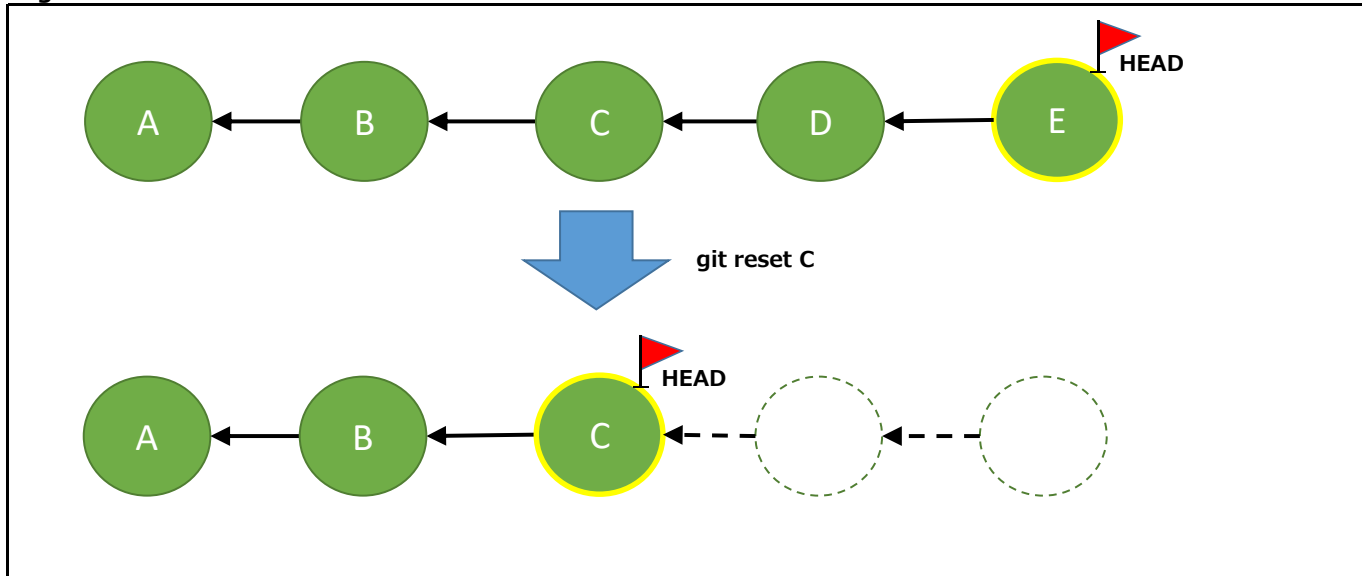


■git reset --hard の動作



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

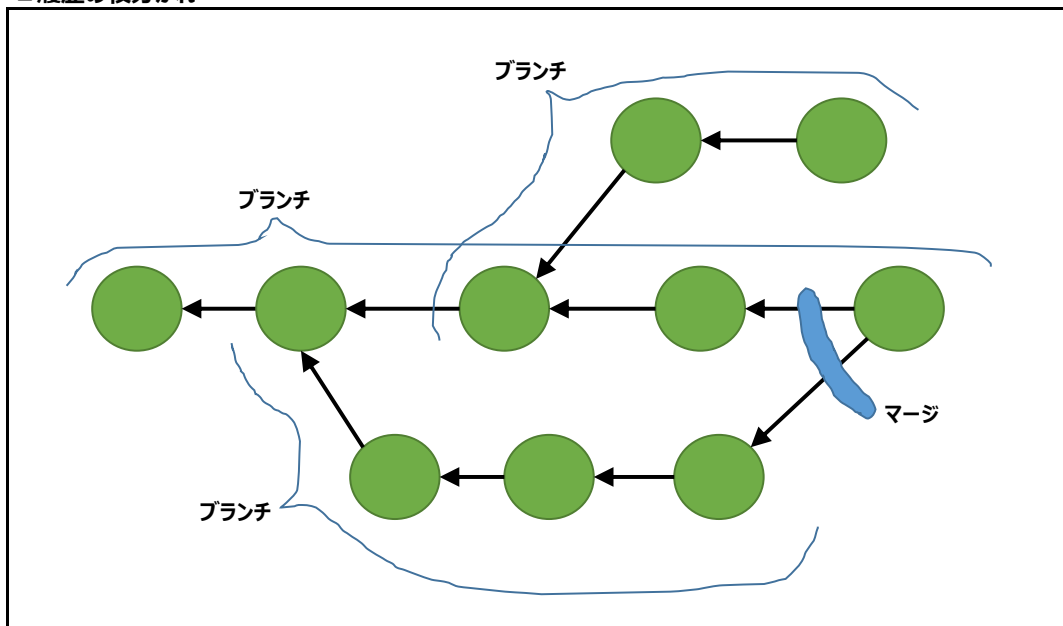


resetコマンドは、HEADの移動を行える。

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）

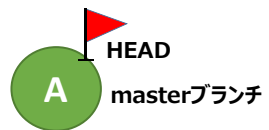
誤って、git resetした後で、HEADを元のコミットに戻す手段もある。

■履歴の枝分かれ

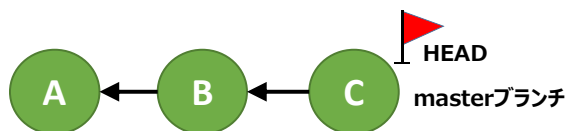


■ ブランチの成長・枝分かれ・切り替え

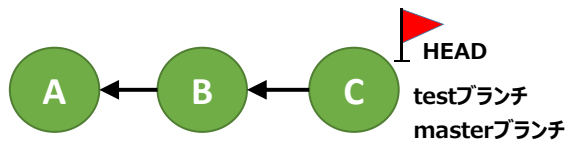
① 最初のコミット



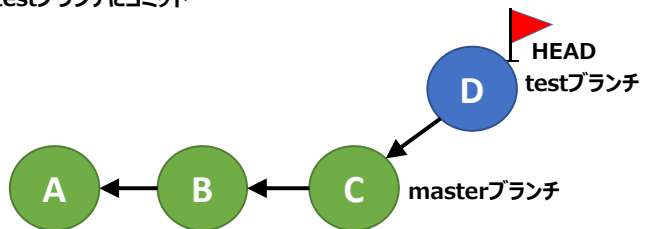
② コミットを2回行う (masterブランチが成長していく)



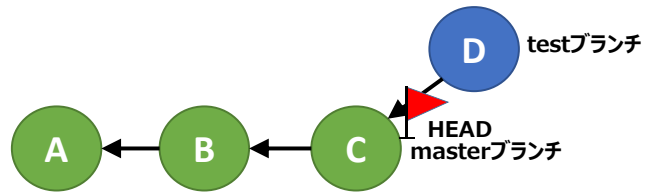
③ testブランチを作成し、HEADを切り替え (下図では分かりづらいが)



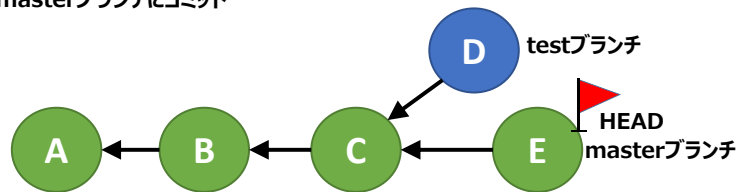
④ testブランチにコミット



⑤ masterブランチに切り替え

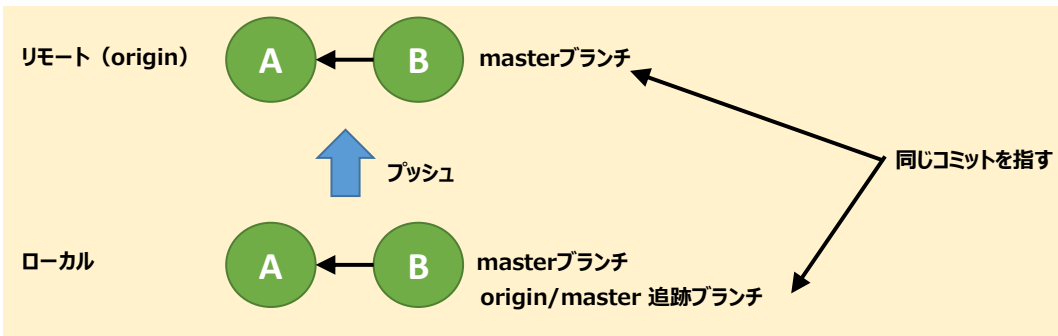


⑥ masterブランチにコミット

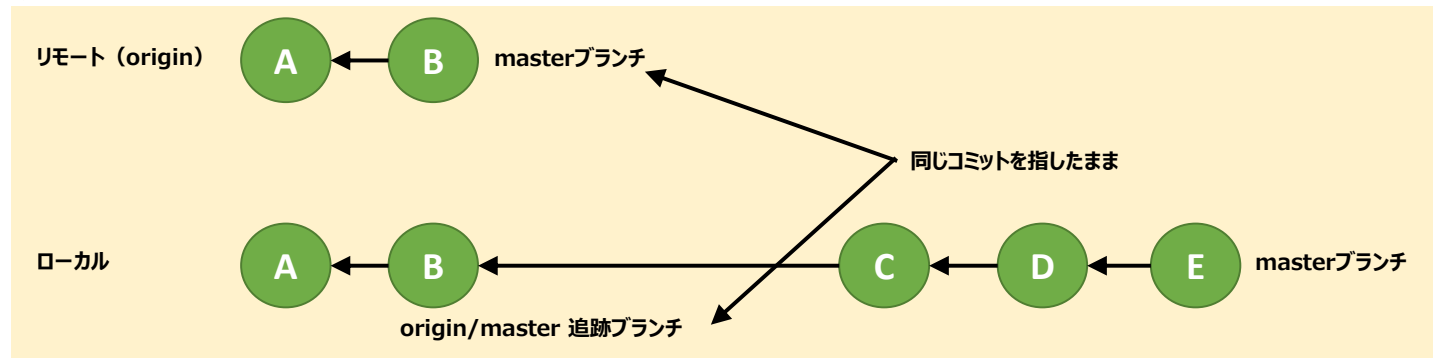


■ 追跡ブランチ

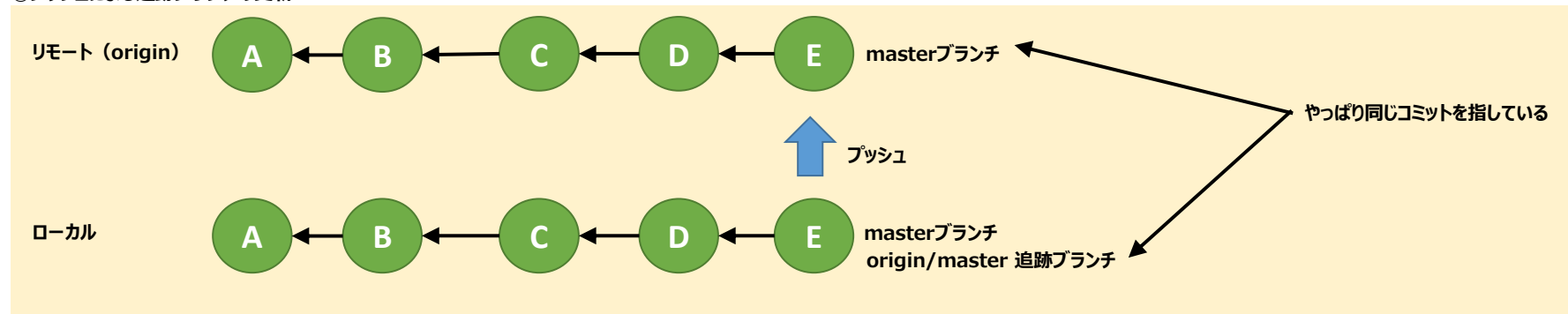
① 追跡ブランチの作成



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 追跡ブランチ
master/ブランチ

