ディレクター・デザイナーのための

git 入門

はじめに

このスライドは今までGitに全く触れたことのない人を対象にしたものです。

概念・感覚を掴むにあたり説明を単純化、デフォルメしているところが多々あります。

Gitがどういうものかを何となく掴めて、もっと深く知りたい、という人は素直にGit Proを読んでください。

用語解説

そもそもGitとは

とてもわかりやすいスライドです。先にこちら↓をご覧ください。

「デザイナのためのGit入門」
https://goo.gl/OIAvVq

おかえりなさい

おかえりなさい。
用語解説＆実際にどう動いているのか？
を解説していきます。

用語解説

リポジトリ

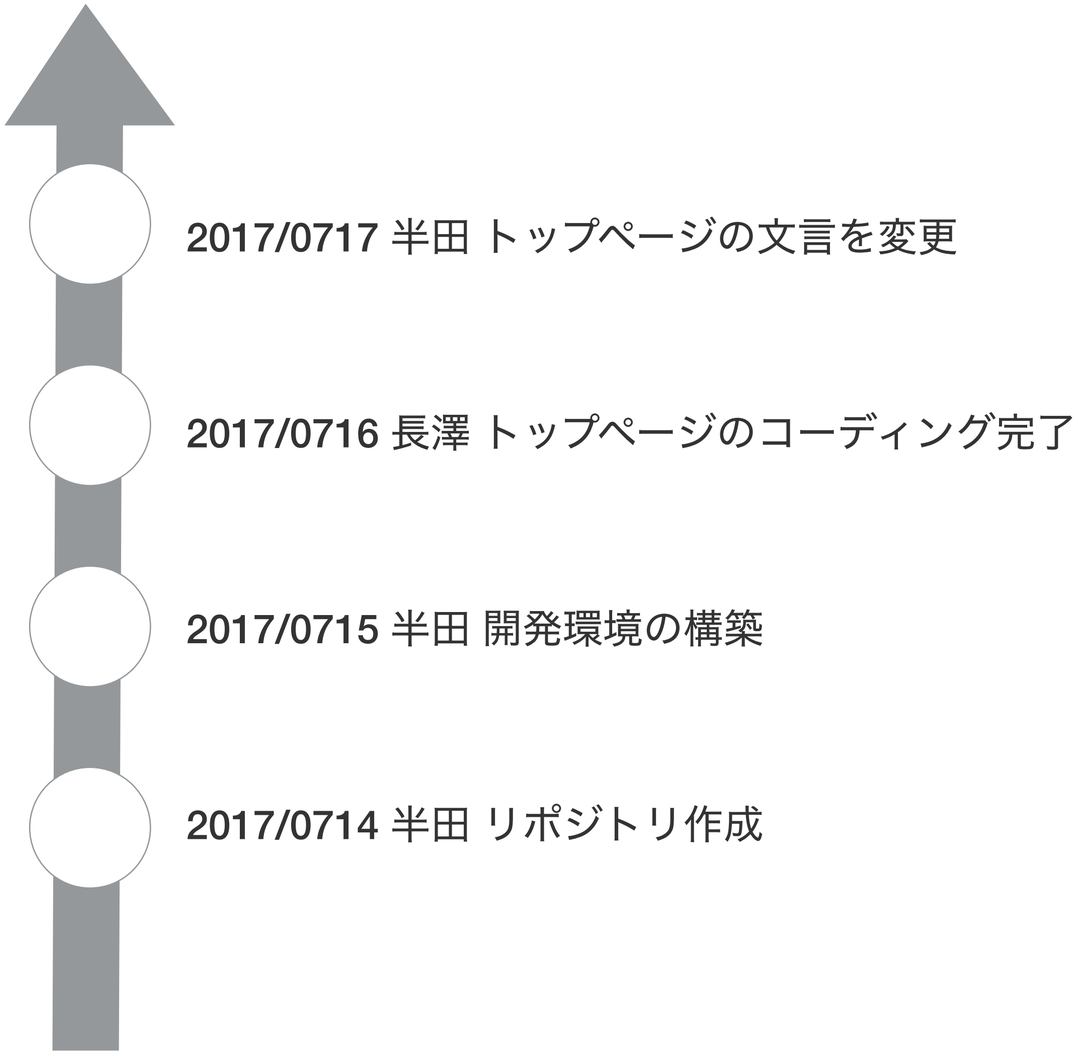
バージョン管理に関する情報（コミット・ブランチ等々）と
実体ファイルが全て格納されている場所。全部。

用語解説

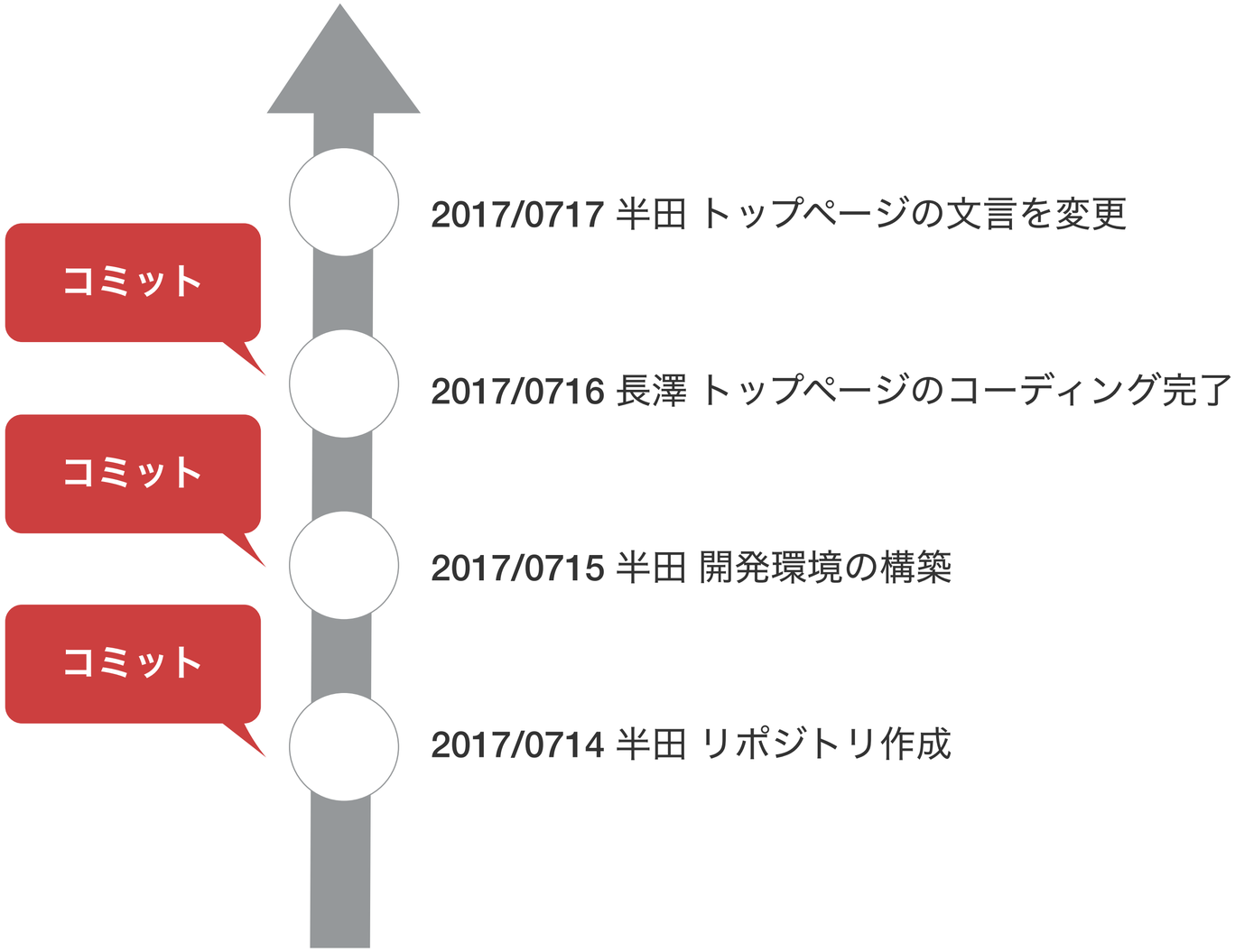
コミット

変更の記録。主に
・ファイルに対する変更記録（どのファイルのどこを変更した）
・それに付随する付加情報（いつ、だれが、作業者のコメント）
を格納している。

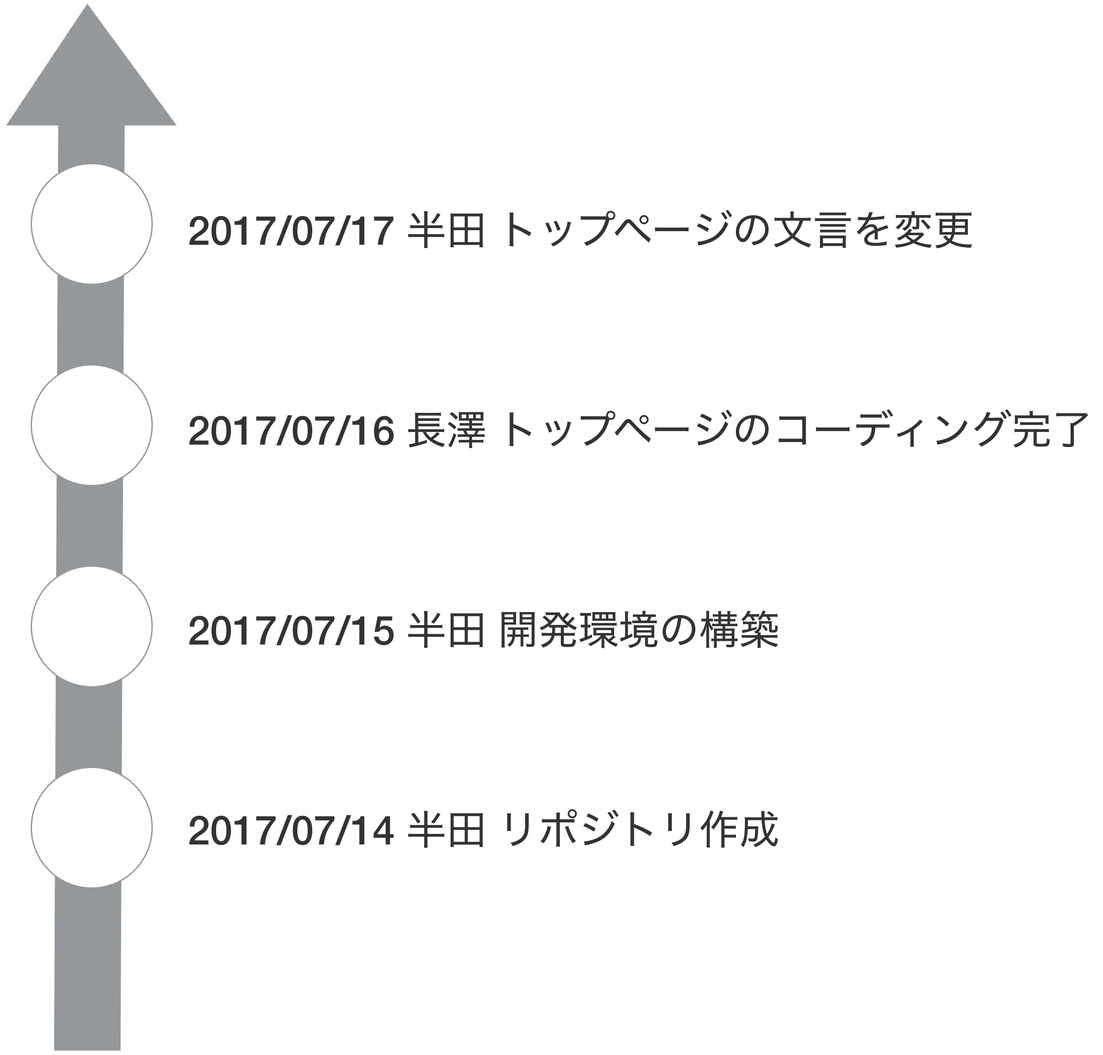
Gitの基本（コミットログ）



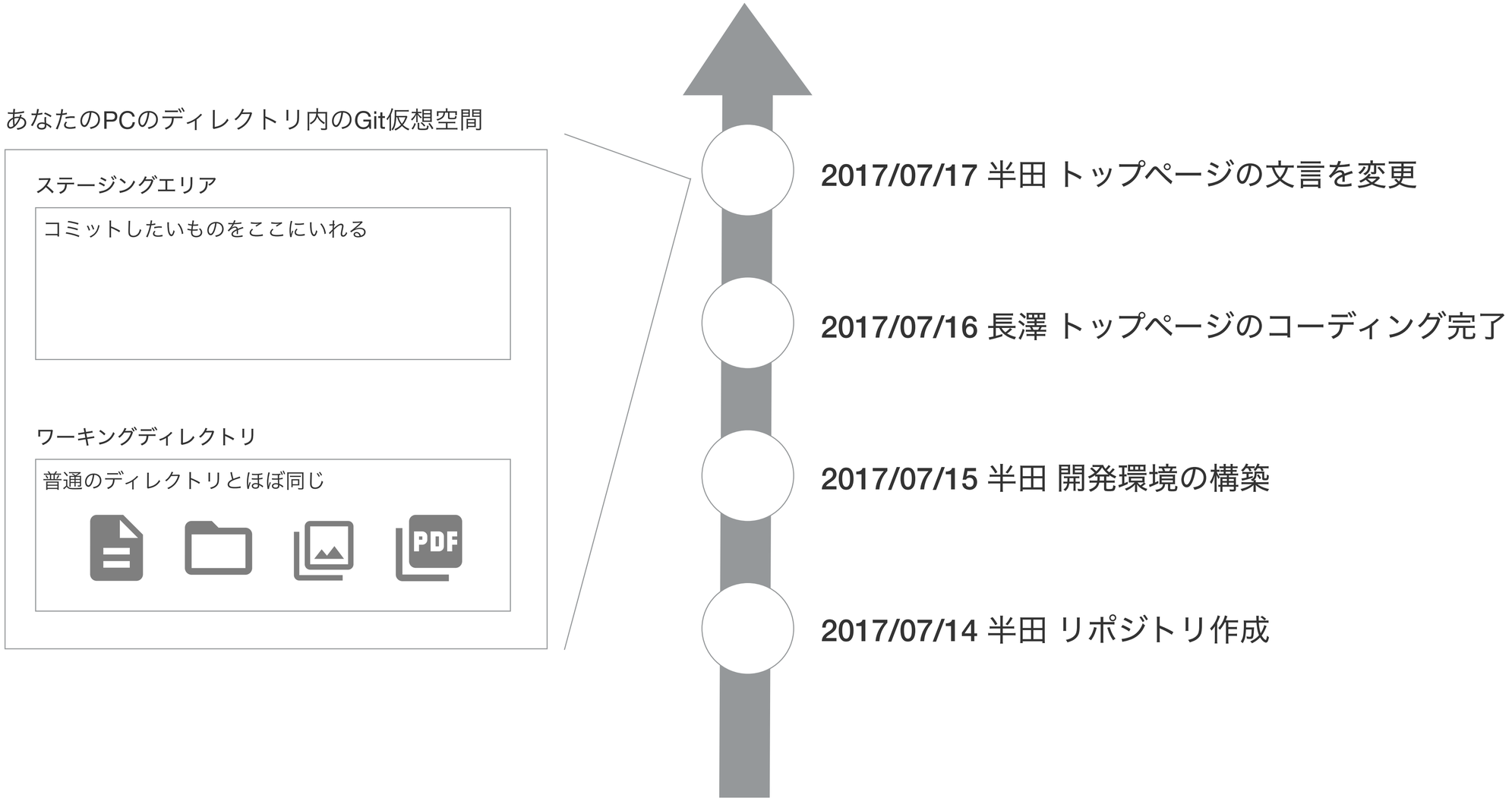
Gitの基本（コミットログ）



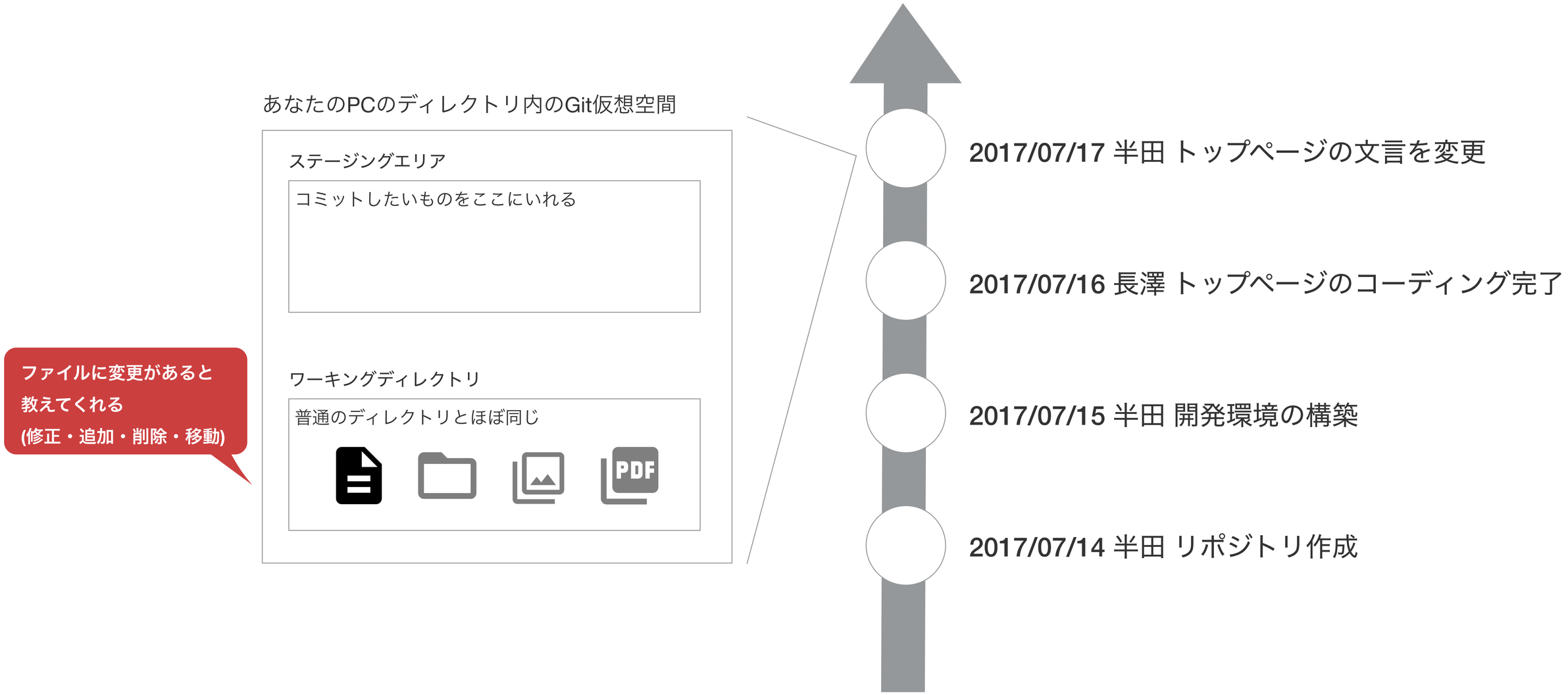
コミットの一連の流れ


2017/07/17 半田 トップページの文言を変更
2017/07/16 長澤 トップページのコーディング完了
2017/07/15 半田 開発環境の構築
2017/07/14 半田 リポジトリ作成

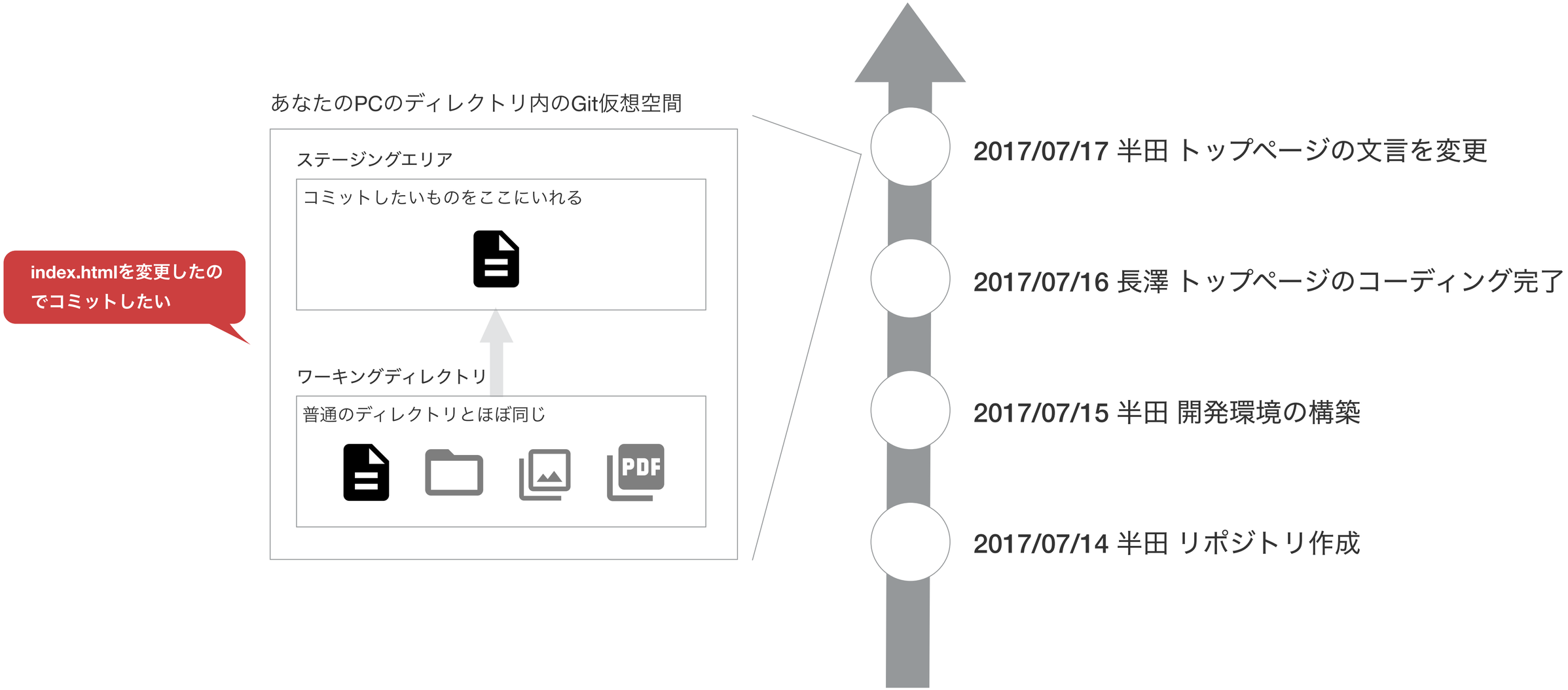
コミットの一連の流れ


あなたのPCのディレクトリ内のGit仮想空間
ステージングエリア
コミットしたいものをここにいれる
ワーキングディレクトリ
普通のディレクトリとほぼ同じ
PDF
2017/07/17 半田 トップページの文言を変更
2017/07/16 長澤 トップページのコーディング完了
2017/07/15 半田 開発環境の構築
2017/07/14 半田 リポジトリ作成

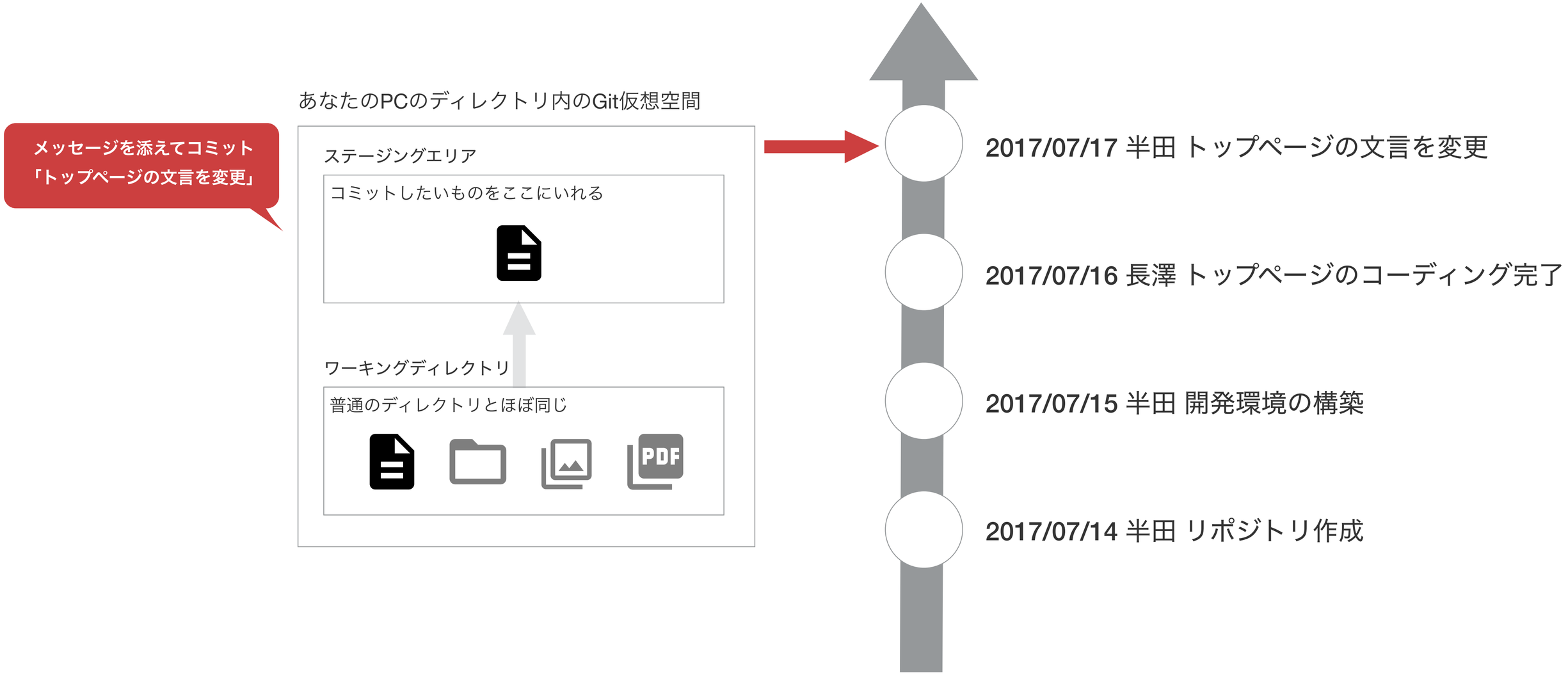
コミットの一連の流れ


あなたのPCのディレクトリ内のGit仮想空間
ステージングエリア
コミットしたいものをここにいれる
ワーキングディレクトリ
普通のディレクトリとほぼ同じ
PDF
ファイルに変更があると
教えてくれる
(修正・追加・削除・移動)
2017/07/17 半田 トップページの文言を変更
2017/07/16 長澤 トップページのコーディング完了
2017/07/15 半田 開発環境の構築
2017/07/14 半田 リポジトリ作成

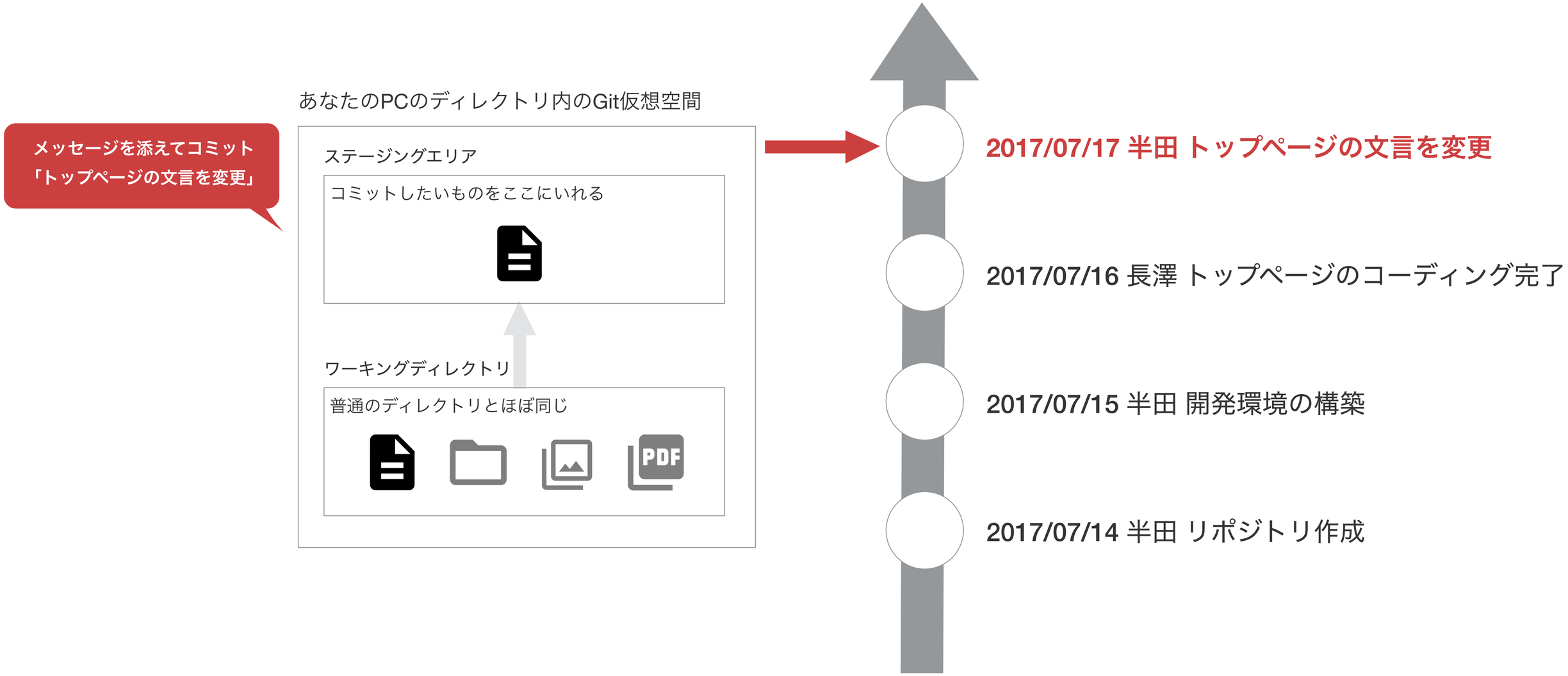
コミットの一連の流れ


index.htmlを変更したのでコミットしたい
あなたのPCのディレクトリ内のGit仮想空間
ステージングエリア
コミットしたいものをここにいれる
ワーキングディレクトリ
普通のディレクトリとほぼ同じ
PDF
2017/07/17 半田 トップページの文言を変更
2017/07/16 長澤 トップページのコーディング完了
2017/07/15 半田 開発環境の構築
2017/07/14 半田 リポジトリ作成

コミットの一連の流れ


メッセージを添えてコミット
「トップページの文言を変更」
あなたのPCのディレクトリ内のGit仮想空間
ステージングエリア
コミットしたいものをここにいれる
ワーキングディレクトリ
普通のディレクトリとほぼ同じ
PDF
2017/07/17 半田 トップページの文言を変更
2017/07/16 長澤 トップページのコーディング完了
2017/07/15 半田 開発環境の構築
2017/07/14 半田 リポジトリ作成

コミットの一連の流れ

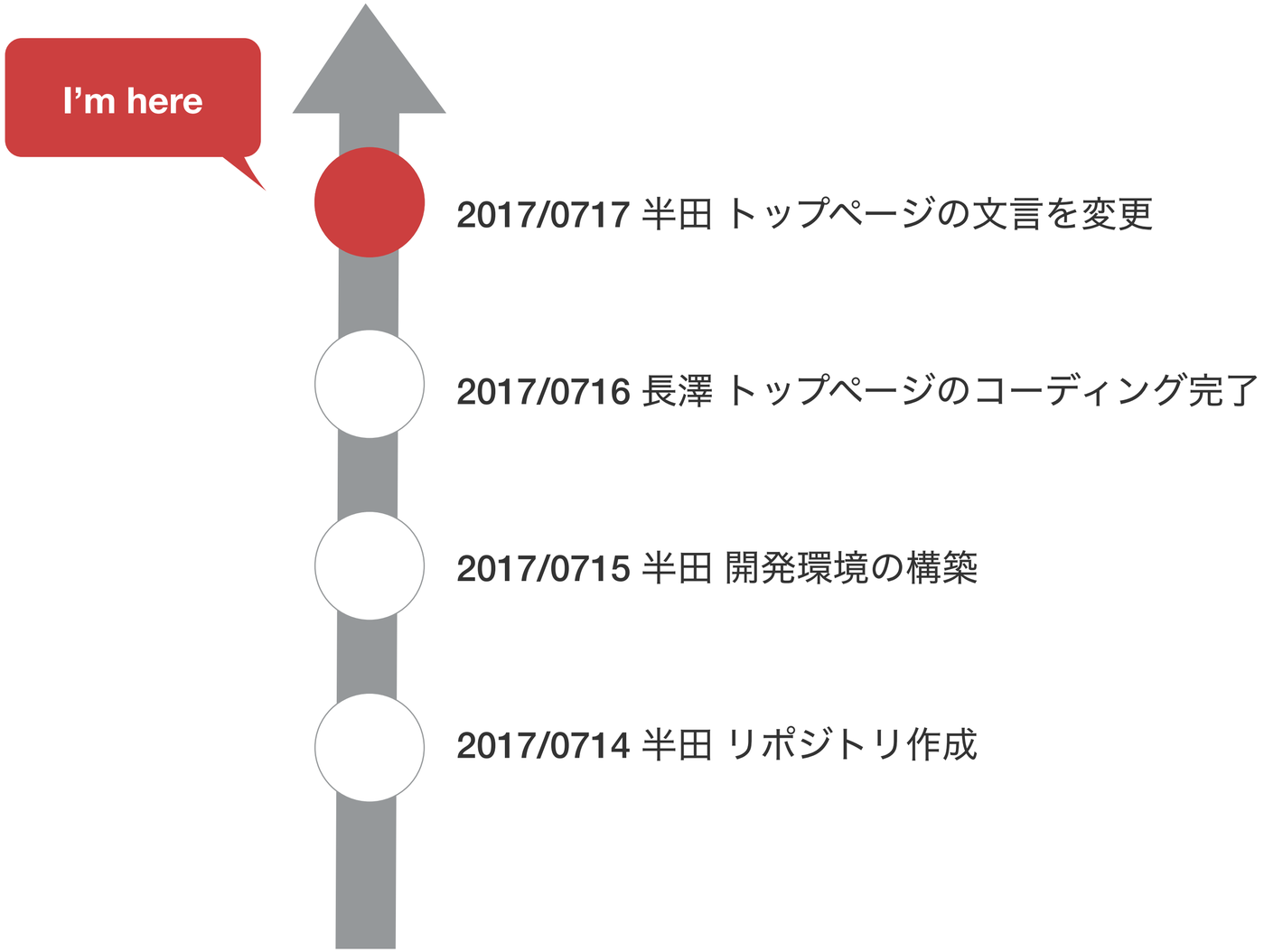

メッセージを添えてコミット
「トップページの文言を変更」
あなたのPCのディレクトリ内のGit仮想空間
ステージングエリア
コミットしたいものをここにいれる
ワーキングディレクトリ
普通のディレクトリとほぼ同じ
PDF
2017/07/17 半田 トップページの文言を変更
2017/07/16 長澤 トップページのコーディング完了
2017/07/15 半田 開発環境の構築
2017/07/14 半田 リポジトリ作成

用語解説

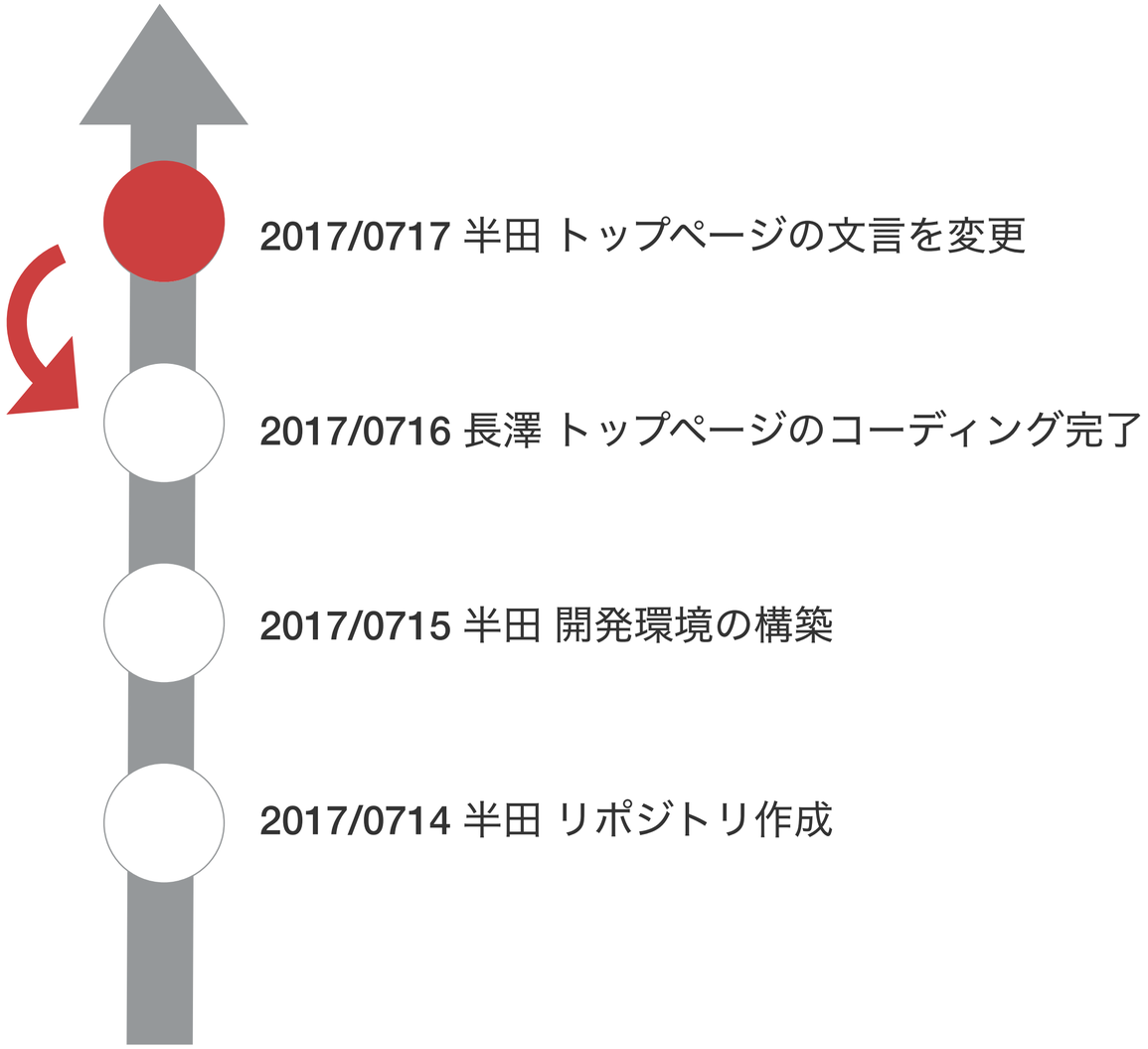
チェックアウト

特定のコミットに切り替えること。

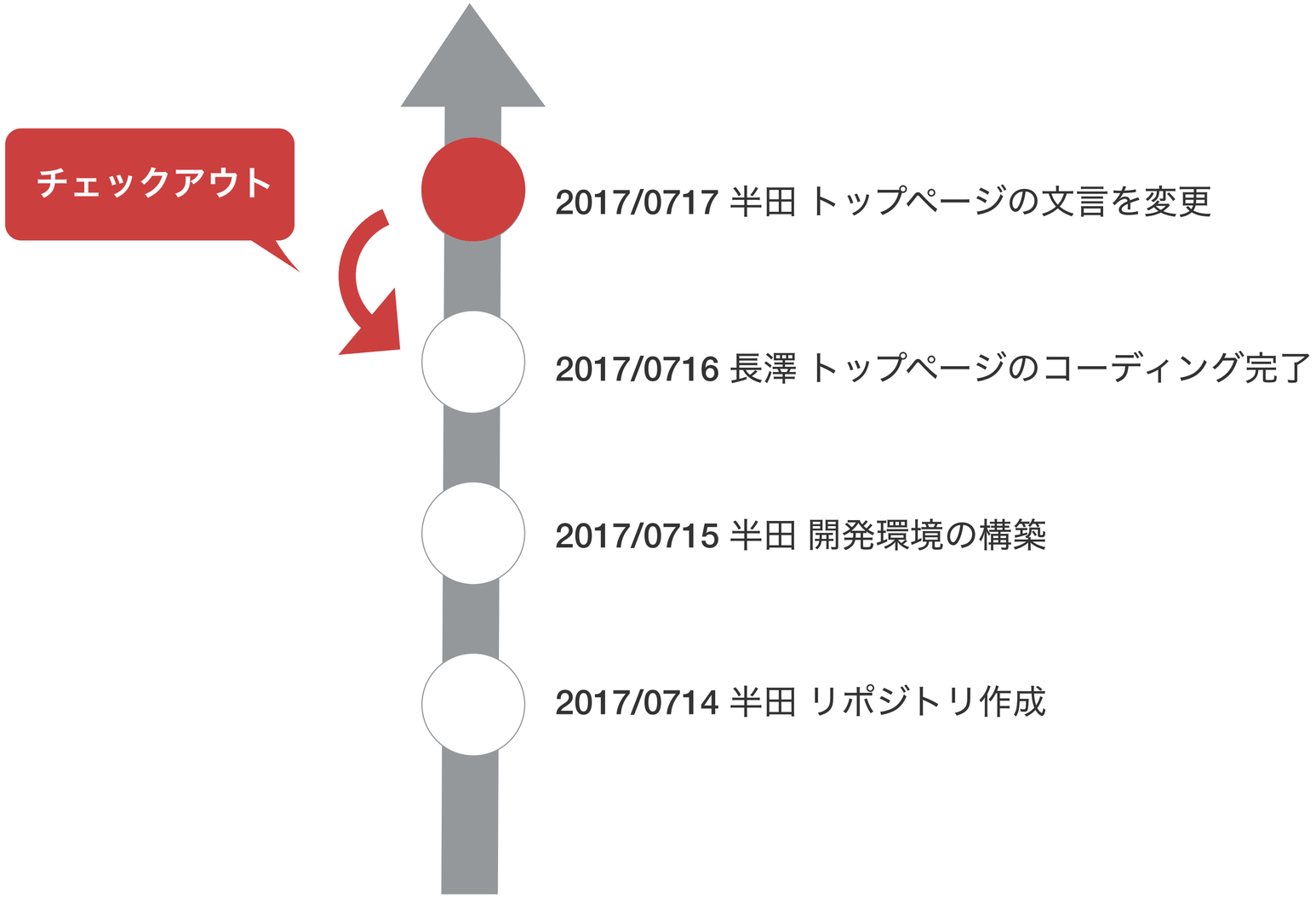
チェックアウト


I’m here
2017/0717 半田 トップページの文言を変更
2017/0716 長澤 トップページのコーディング完了
2017/0715 半田 開発環境の構築
2017/0714 半田 リポジトリ作成

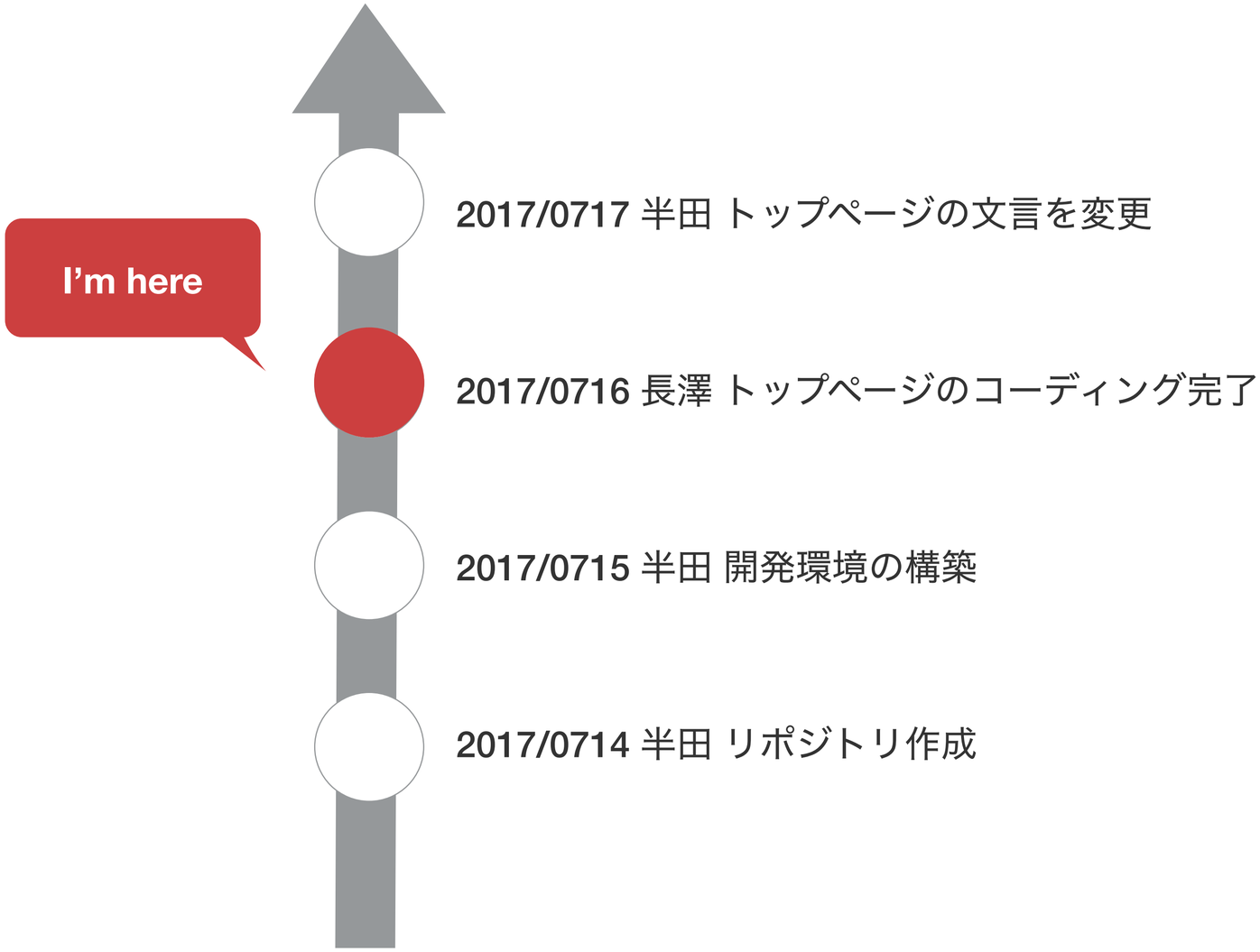
チェックアウト


2017/0717 半田 トップページの文言を変更
2017/0716 長澤 トップページのコーディング完了
2017/0715 半田 開発環境の構築
2017/0714 半田 リポジトリ作成

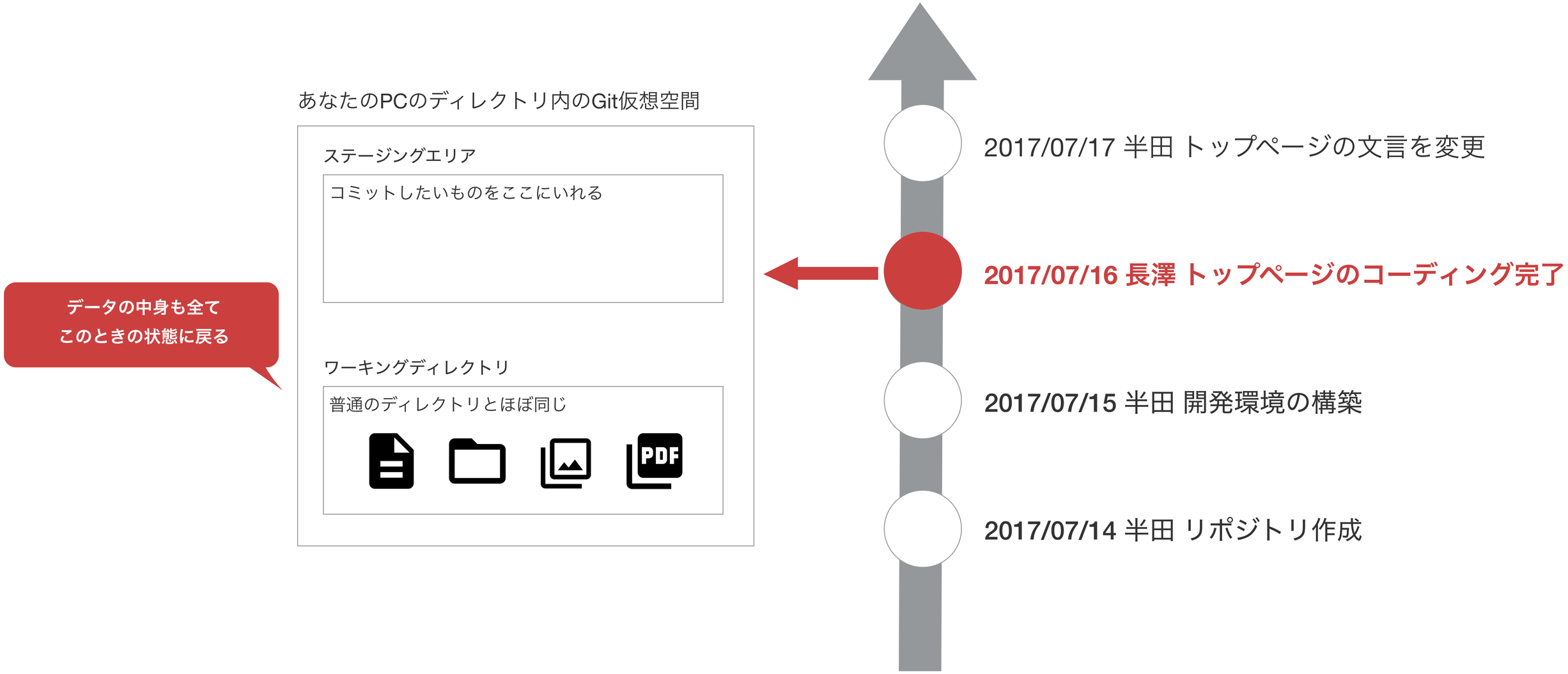
チェックアウト


チェックアウト
2017/0717 半田 トップページの文言を変更
2017/0716 長澤 トップページのコーディング完了
2017/0715 半田 開発環境の構築
2017/0714 半田 リポジトリ作成

チェックアウト


I’m here
2017/0717 半田 トップページの文言を変更
2017/0716 長澤 トップページのコーディング完了
2017/0715 半田 開発環境の構築
2017/0714 半田 リポジトリ作成

チェックアウト


あなたのPCのディレクトリ内のGit仮想空間
ステージングエリア
コミットしたいものをここにいれる
ワーキングディレクトリ
普通のディレクトリとほぼ同じ
PDF
データの中身も全て
このときの状態に戻る
2017/07/17 半田 トップページの文言を変更
2017/07/16 長澤 トップページのコーディング完了
2017/07/15 半田 開発環境の構築
2017/07/14 半田 リポジトリ作成

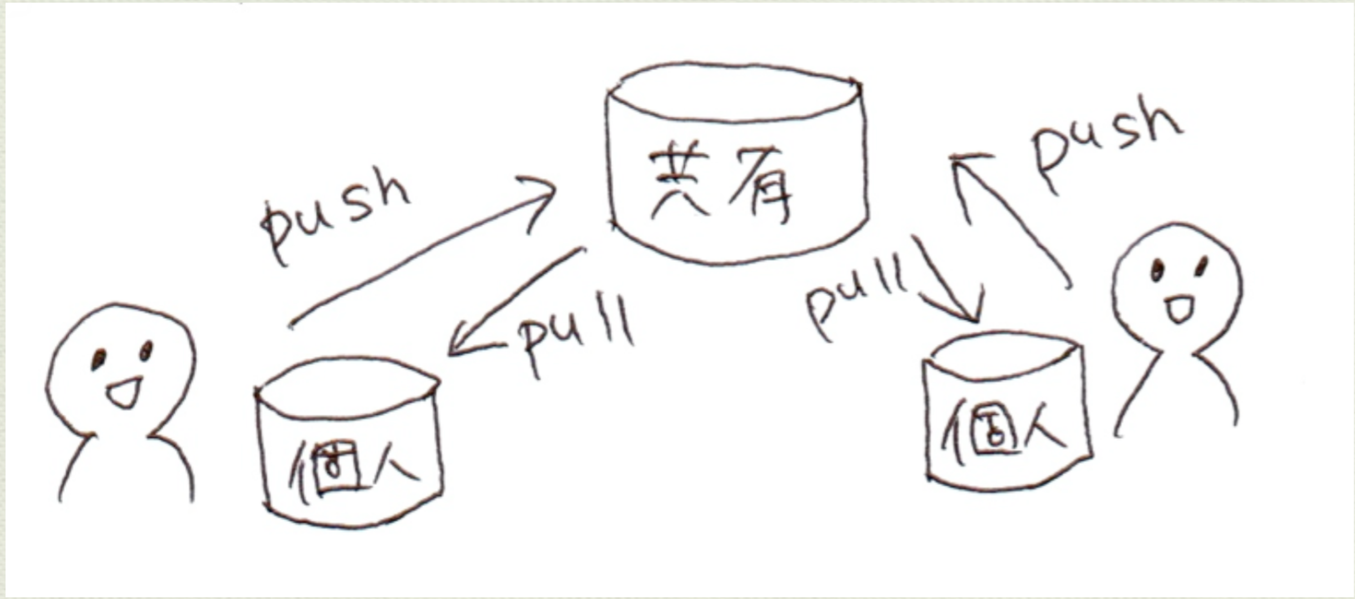
用語解説

リモートリポジトリ

今までのリポジトリ（ローカルリポジトリ）と大して変わらないです。サーバー上にあるだけです。
それによって、他の人との連携がしやすくなります。
逆に言うと、今までのようにリモートリポジトリ無しに自分のPC内だけでGitすることもできます。

この感じ

Gitで集中管理ぽく

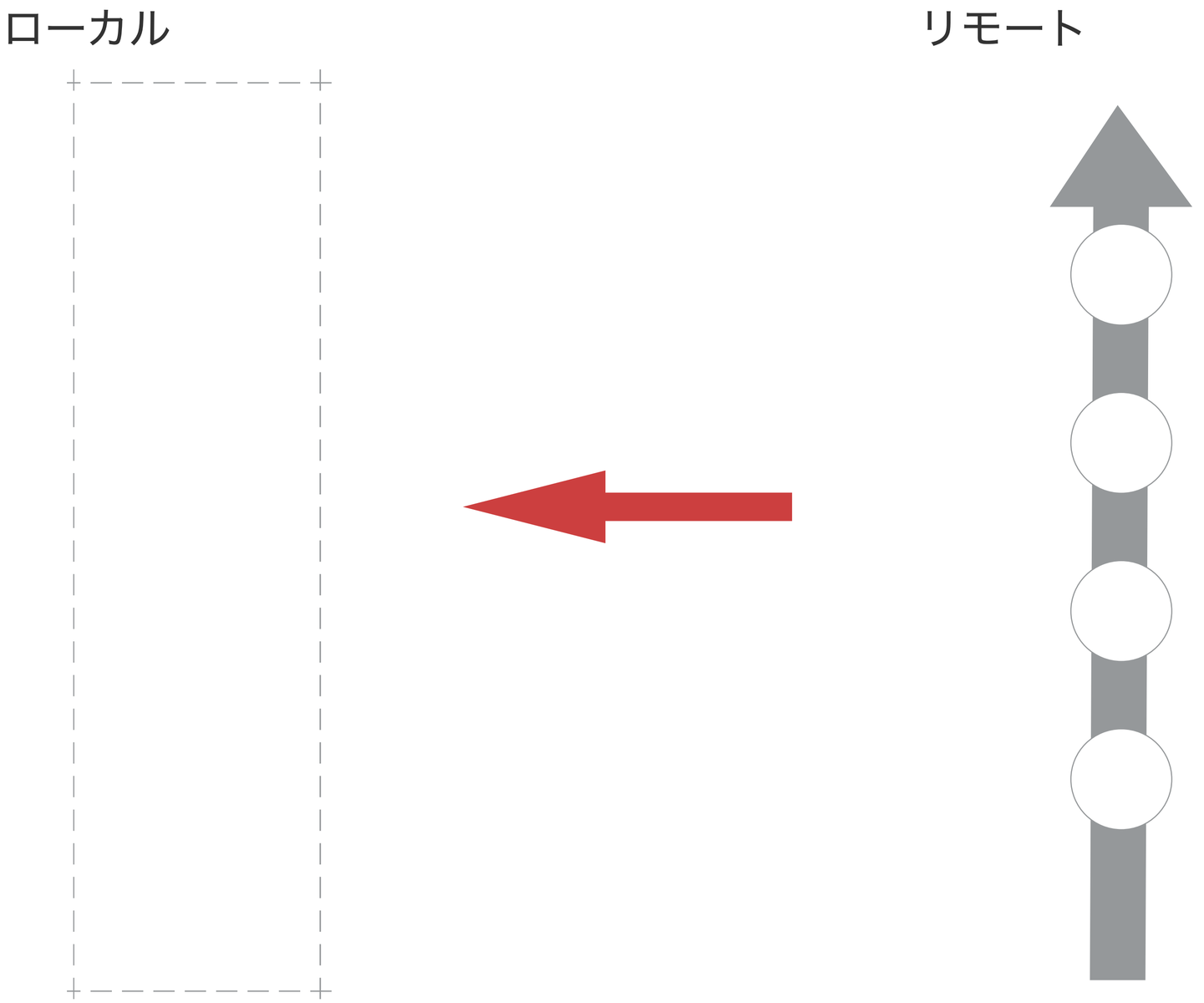


- 個人のリポジトリと共有リポジトリを使う
- 共有リポジトリへpush/pullして、同期する。

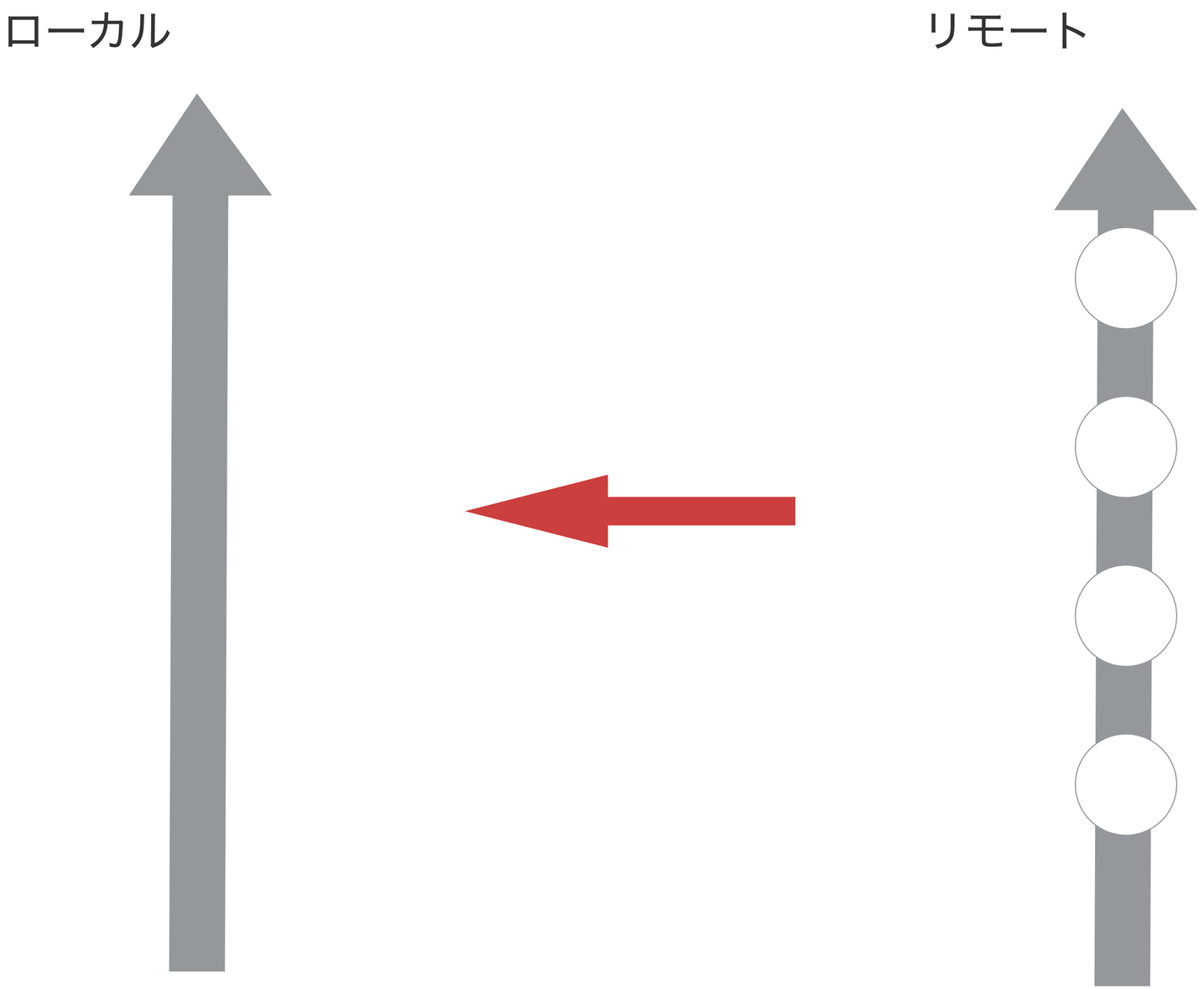
クローン

既存のリポジトリを取得してローカルリポジトリを作成すること。
SVNでいうチェックアウト。

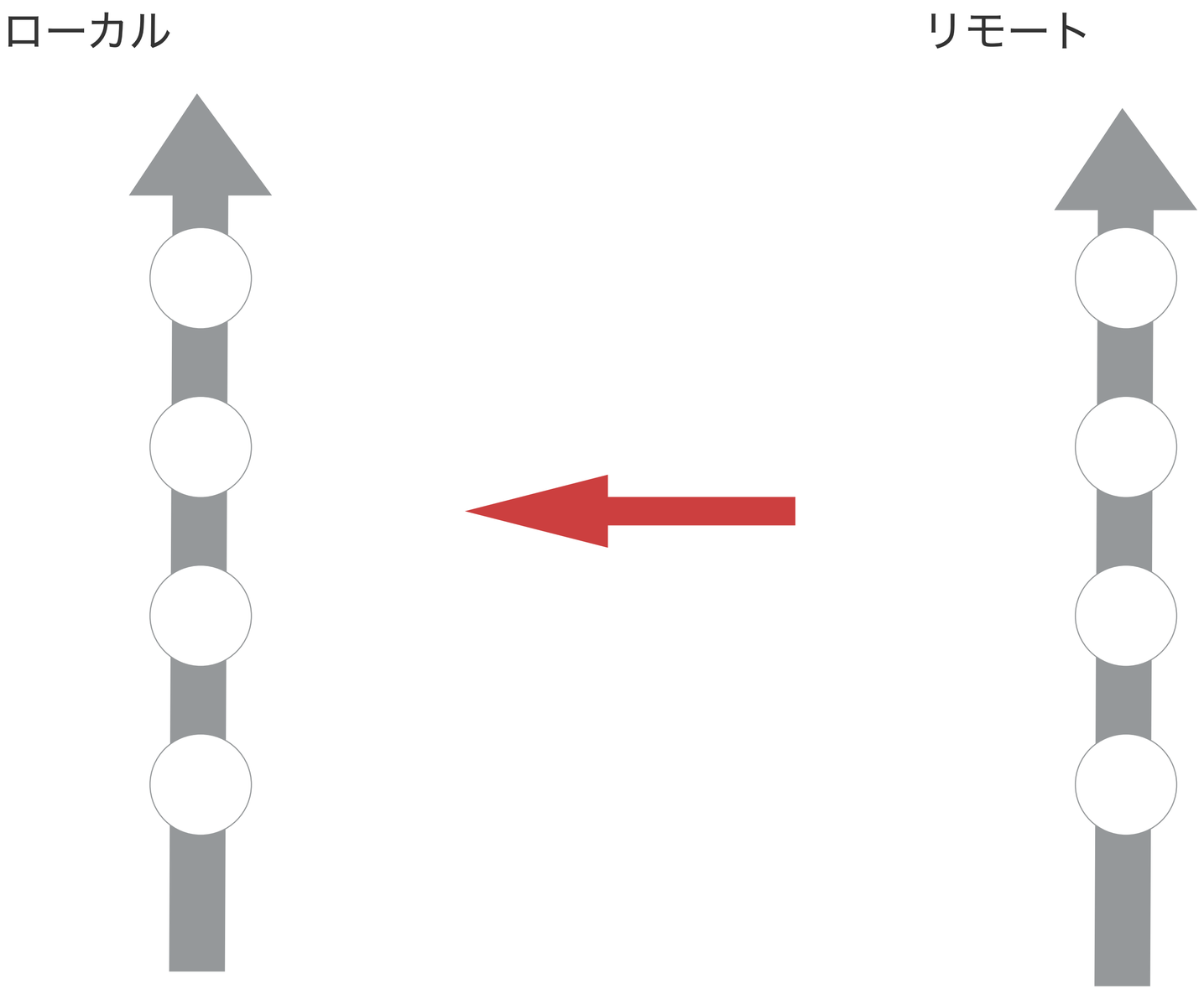
クローン



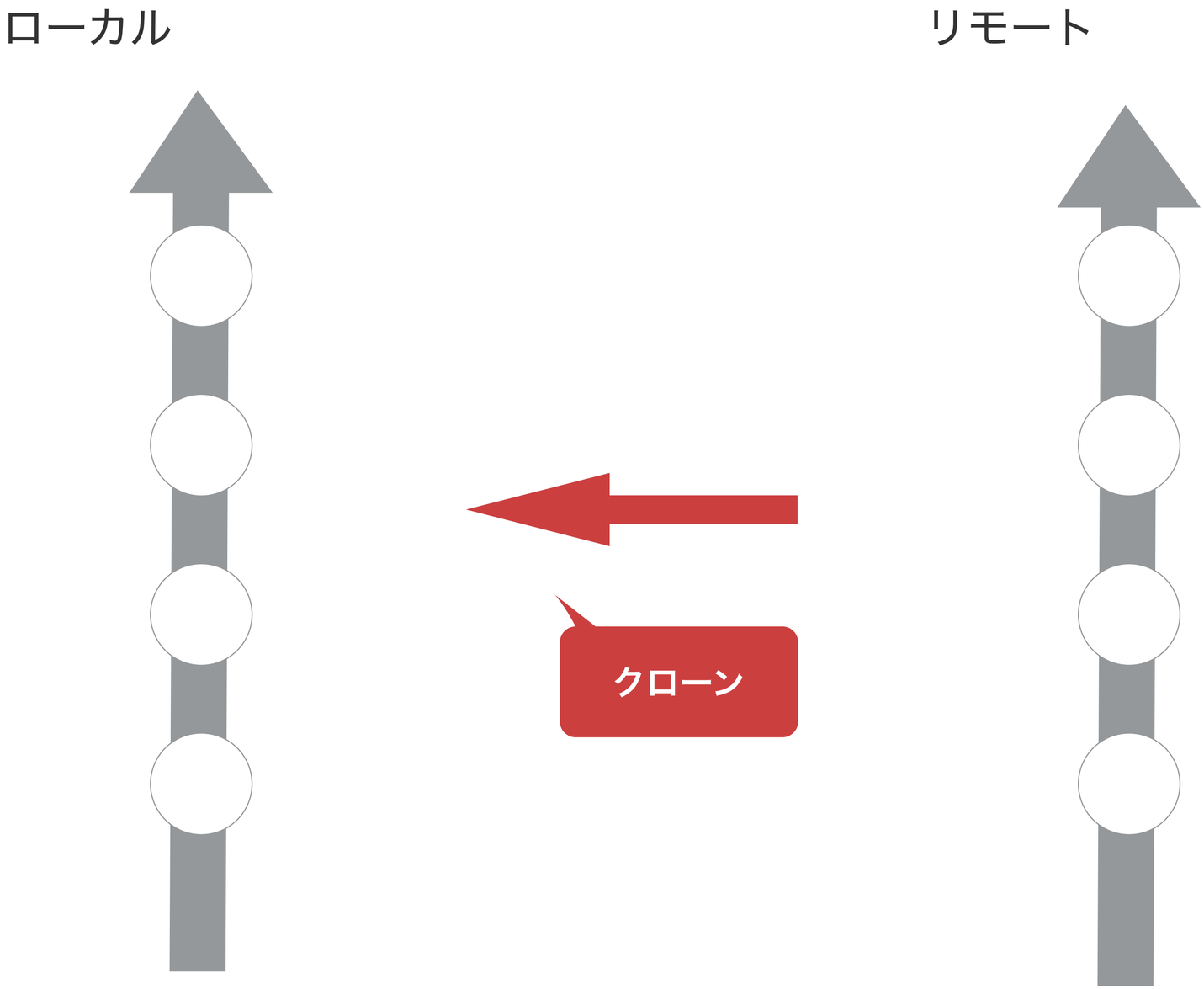
クローン



クローン



クローン

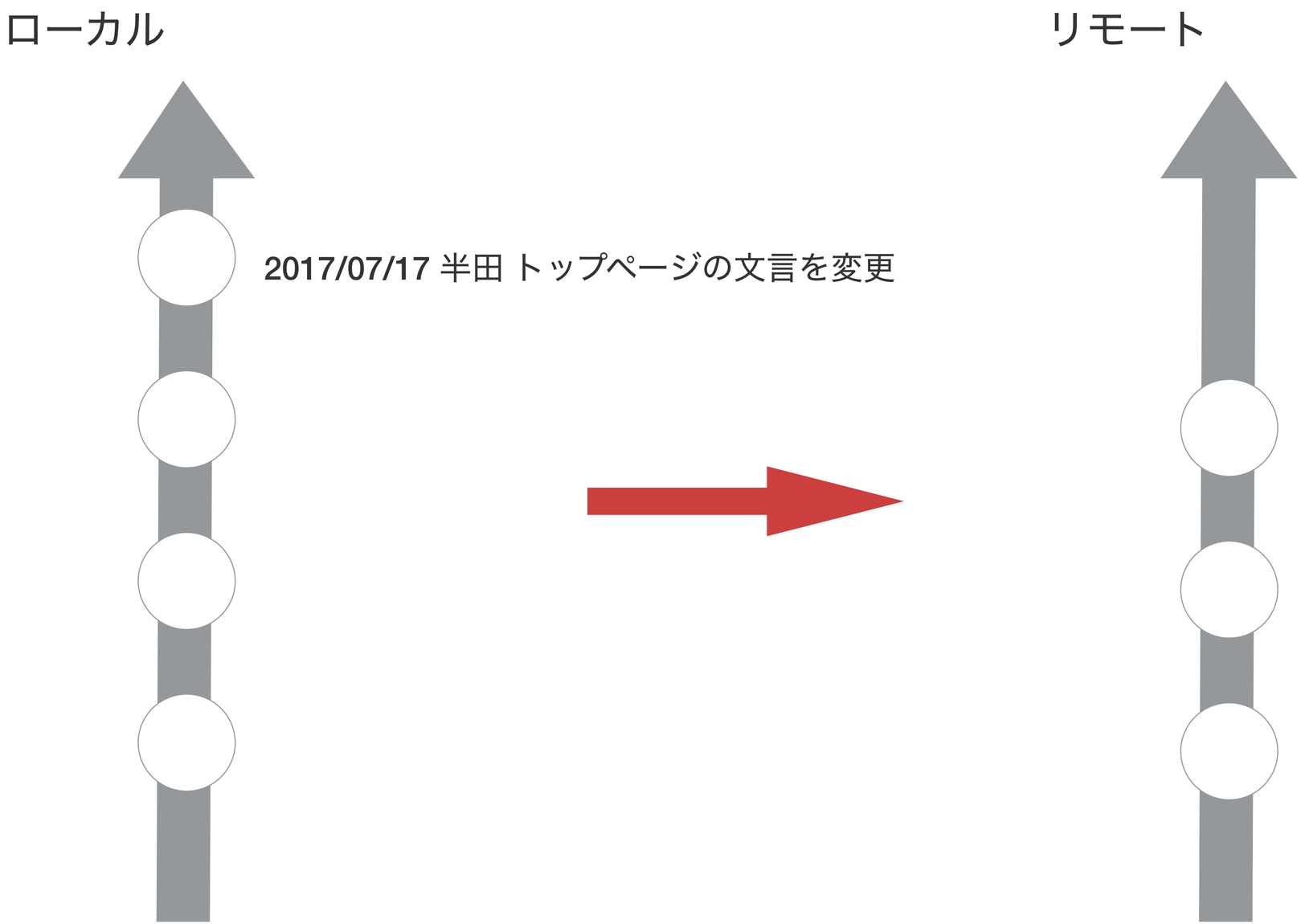


用語解説

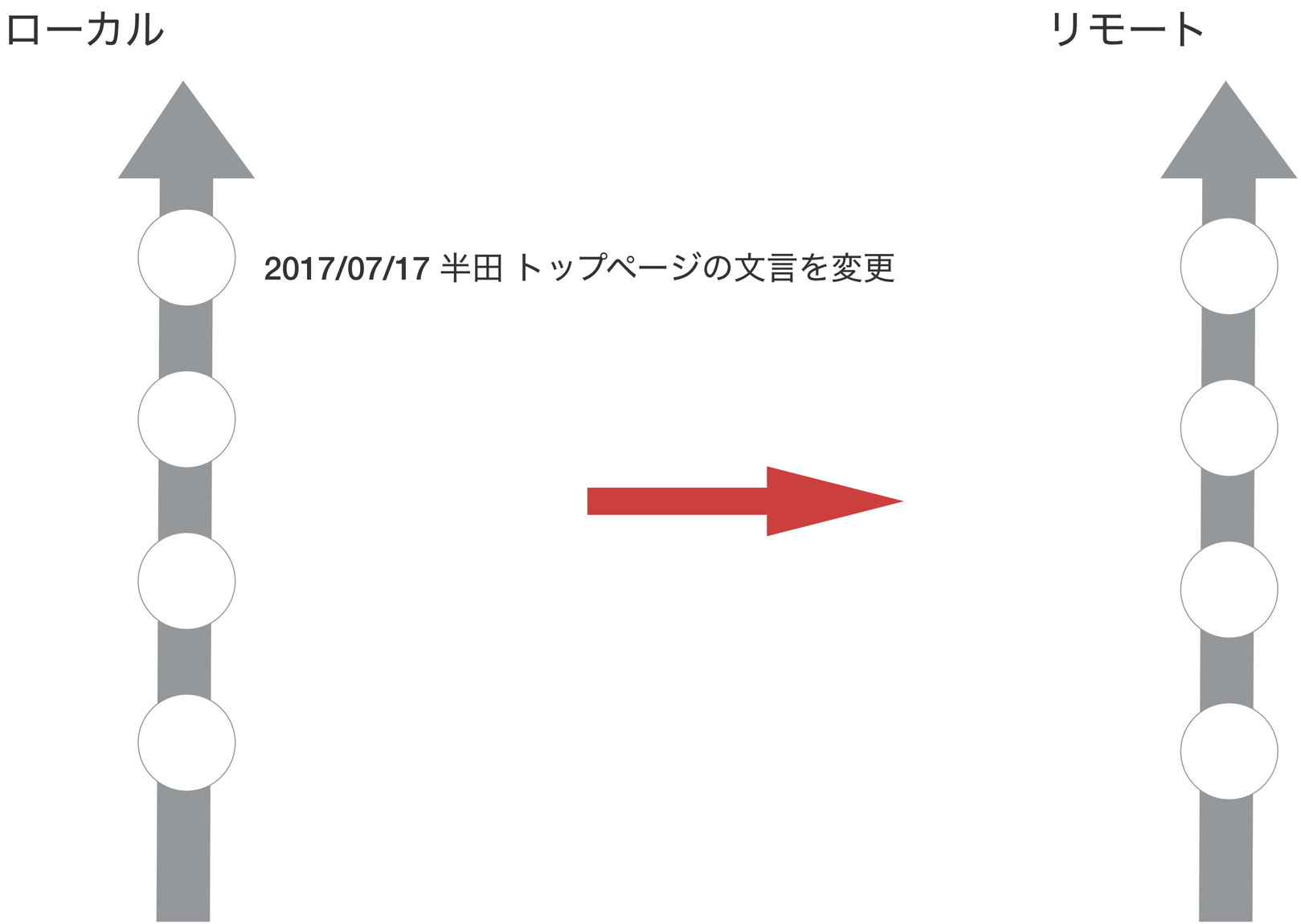
プッシュ

ローカルリポジトリの情報をリモートリポジトリに反映すること。

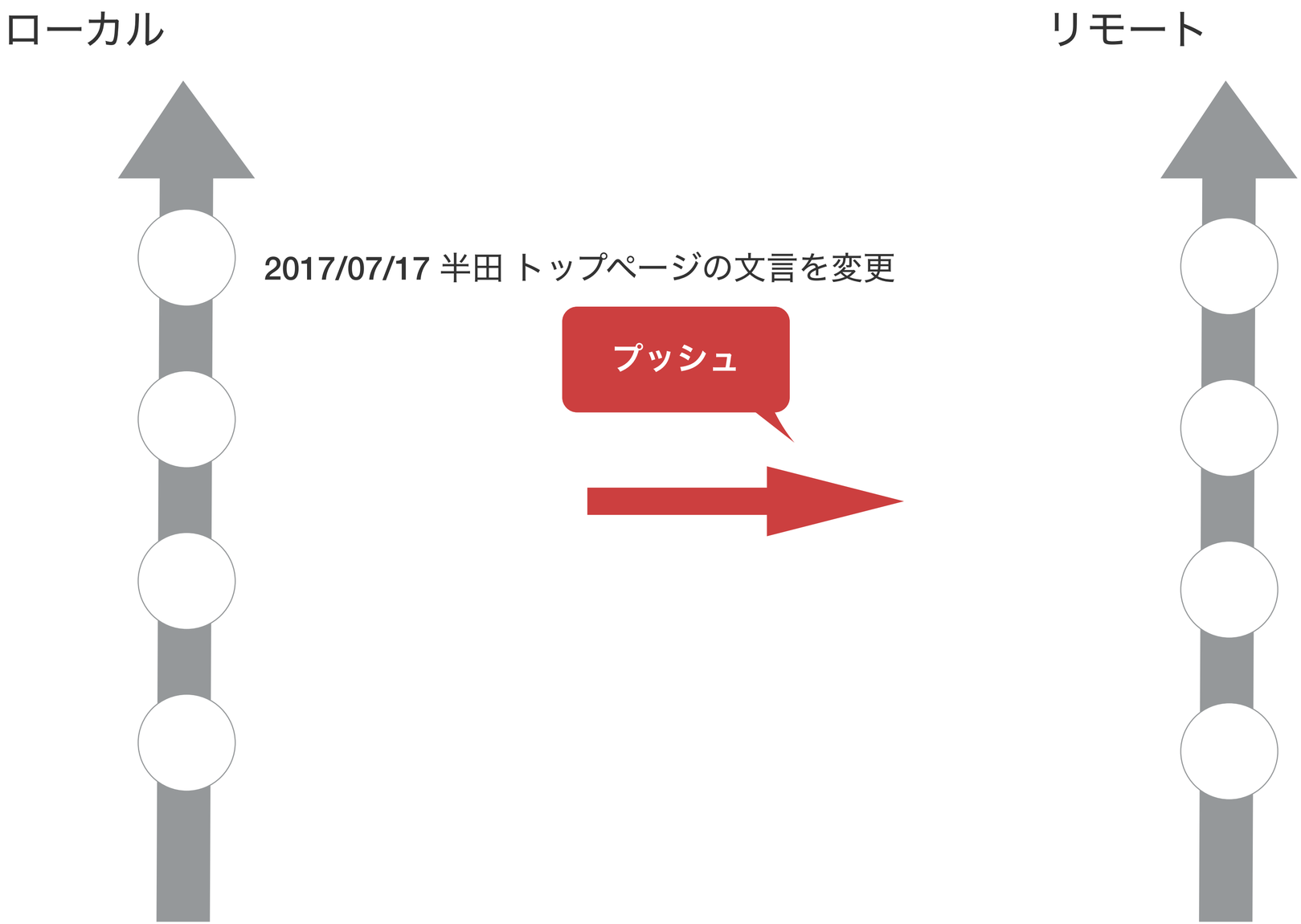
プッシュ



プッシュ



プッシュ

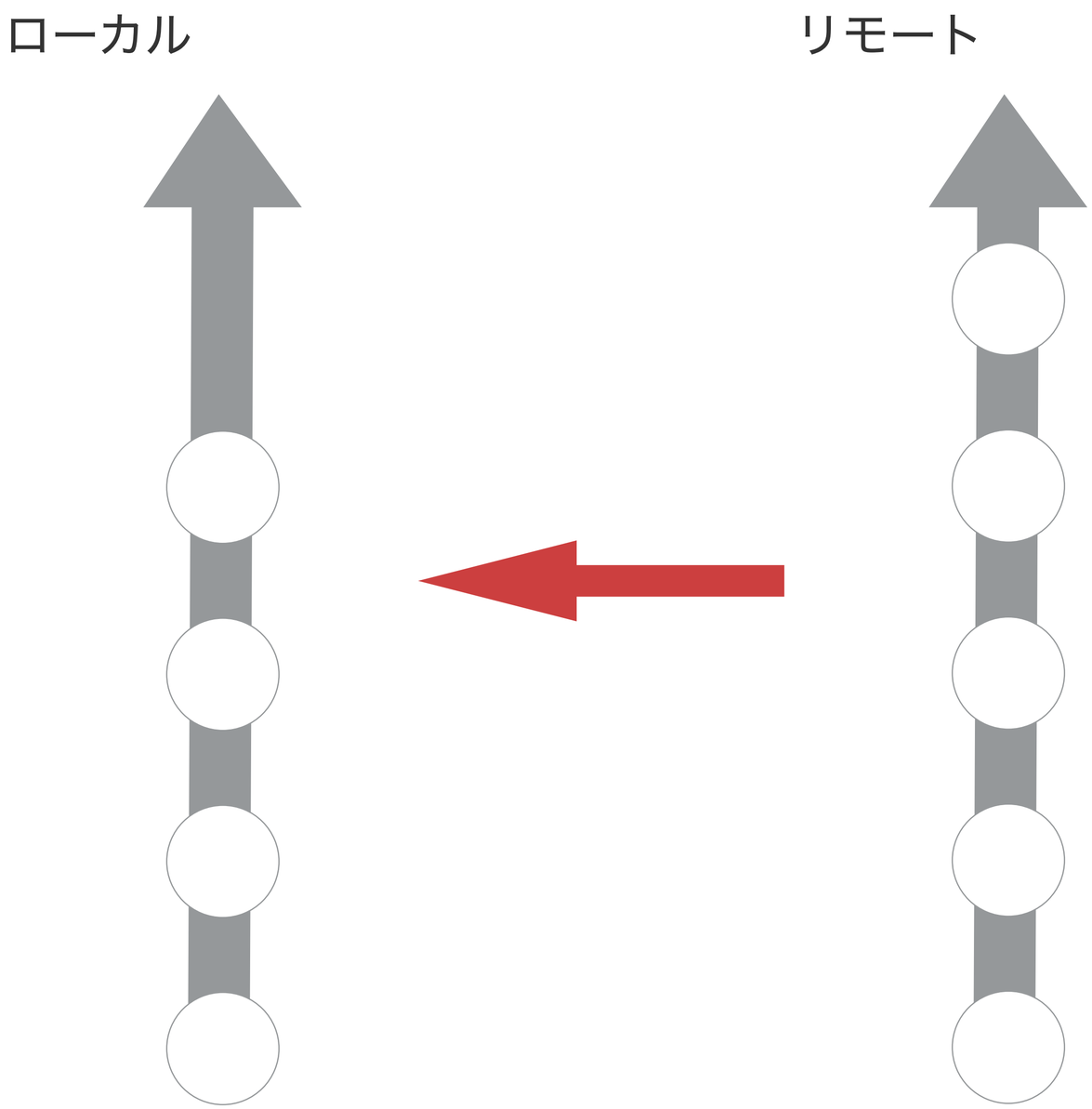


用語解説

プル

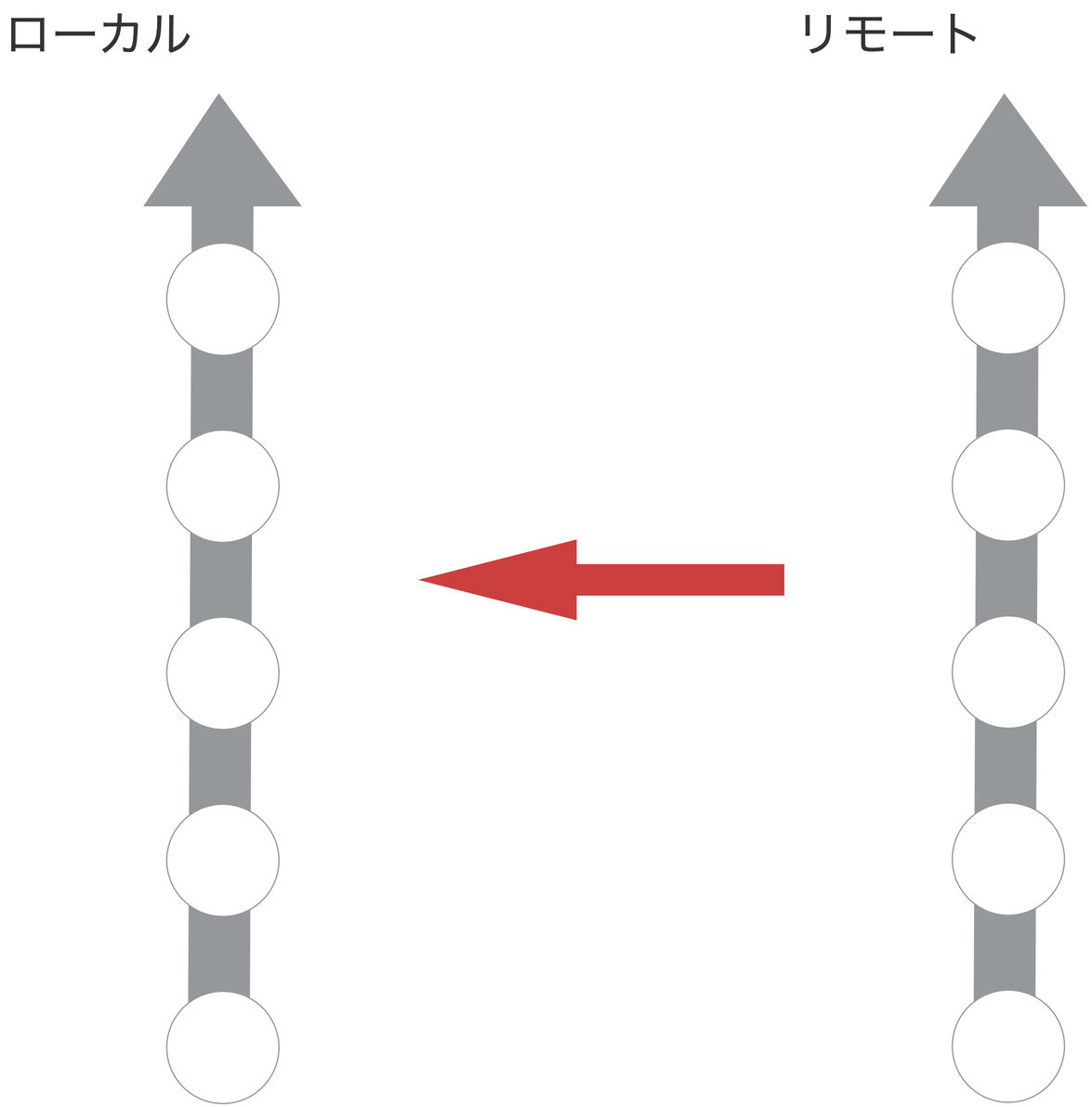
リモートリポジトリの情報をローカルリポジトリに反映すること。
（フェッチ＋マージですが、覚えなくていいです）

プル



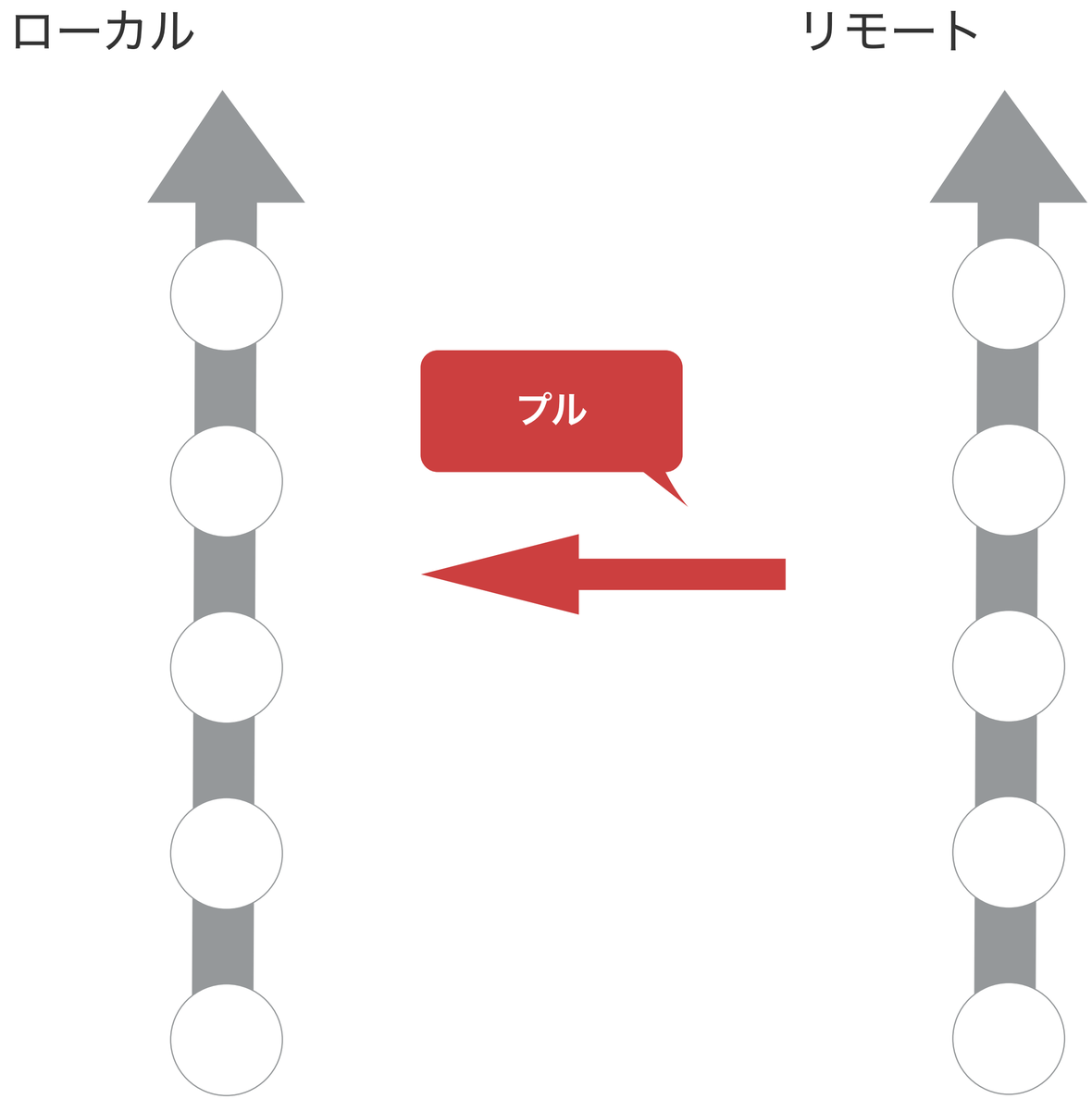
2017/07/18 長澤 下層ページの作成

プル



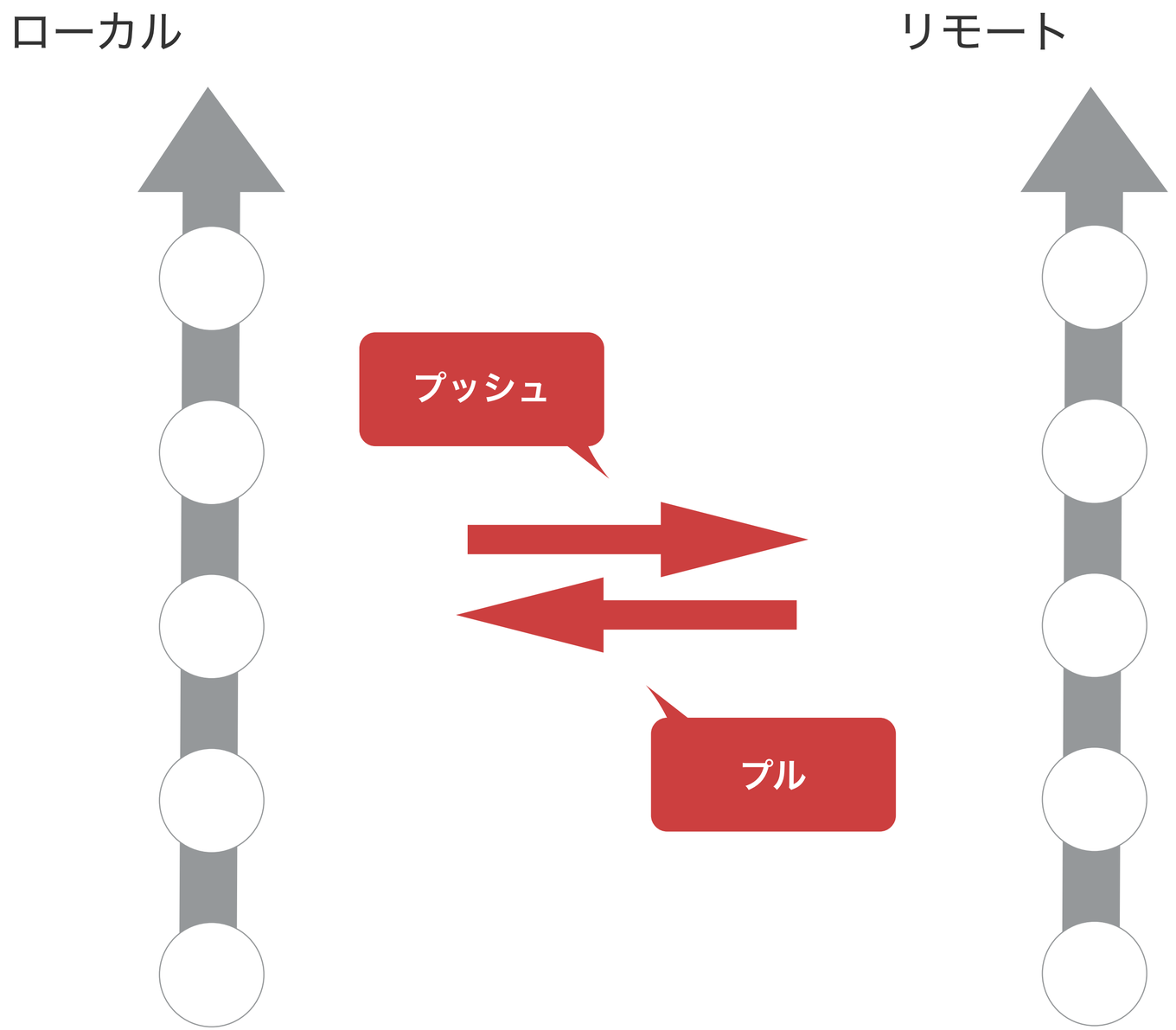
2017/07/18 長澤 下層ページの作成

プル

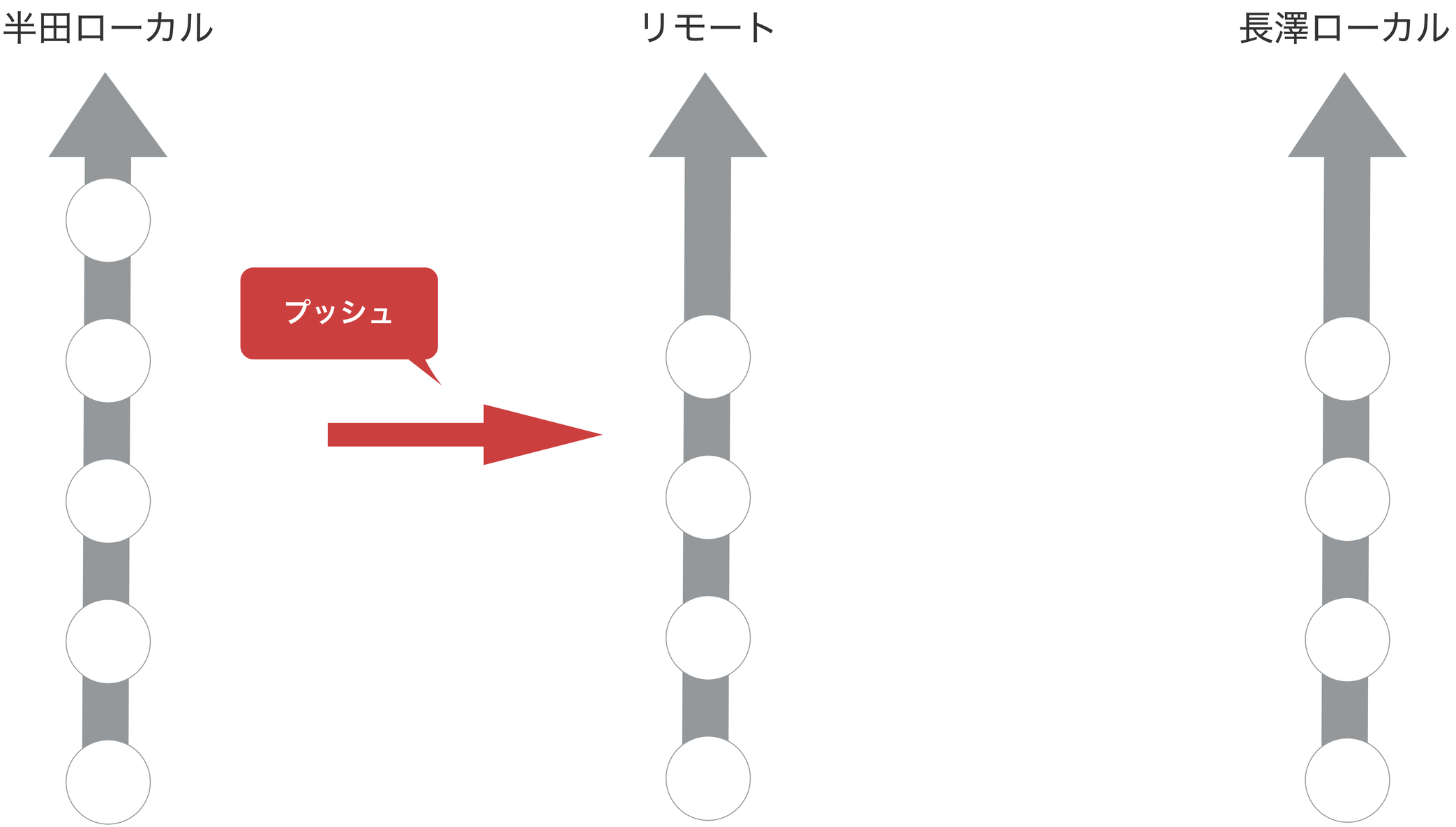


2017/07/18 長澤 下層ページの作成

プッシュ＆プル

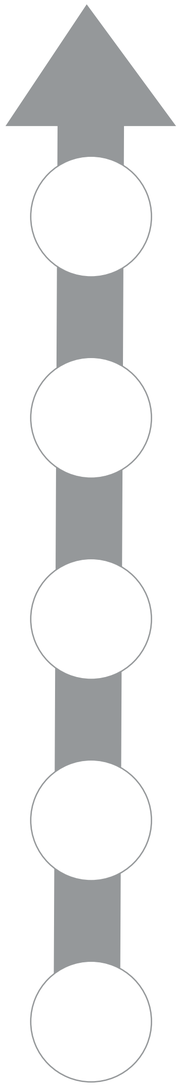


複数人での作業

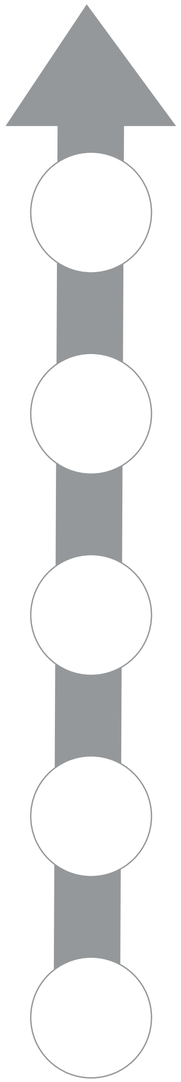


複数人での作業

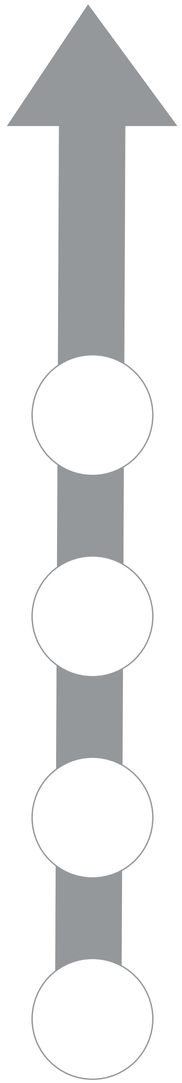
半田ローカル



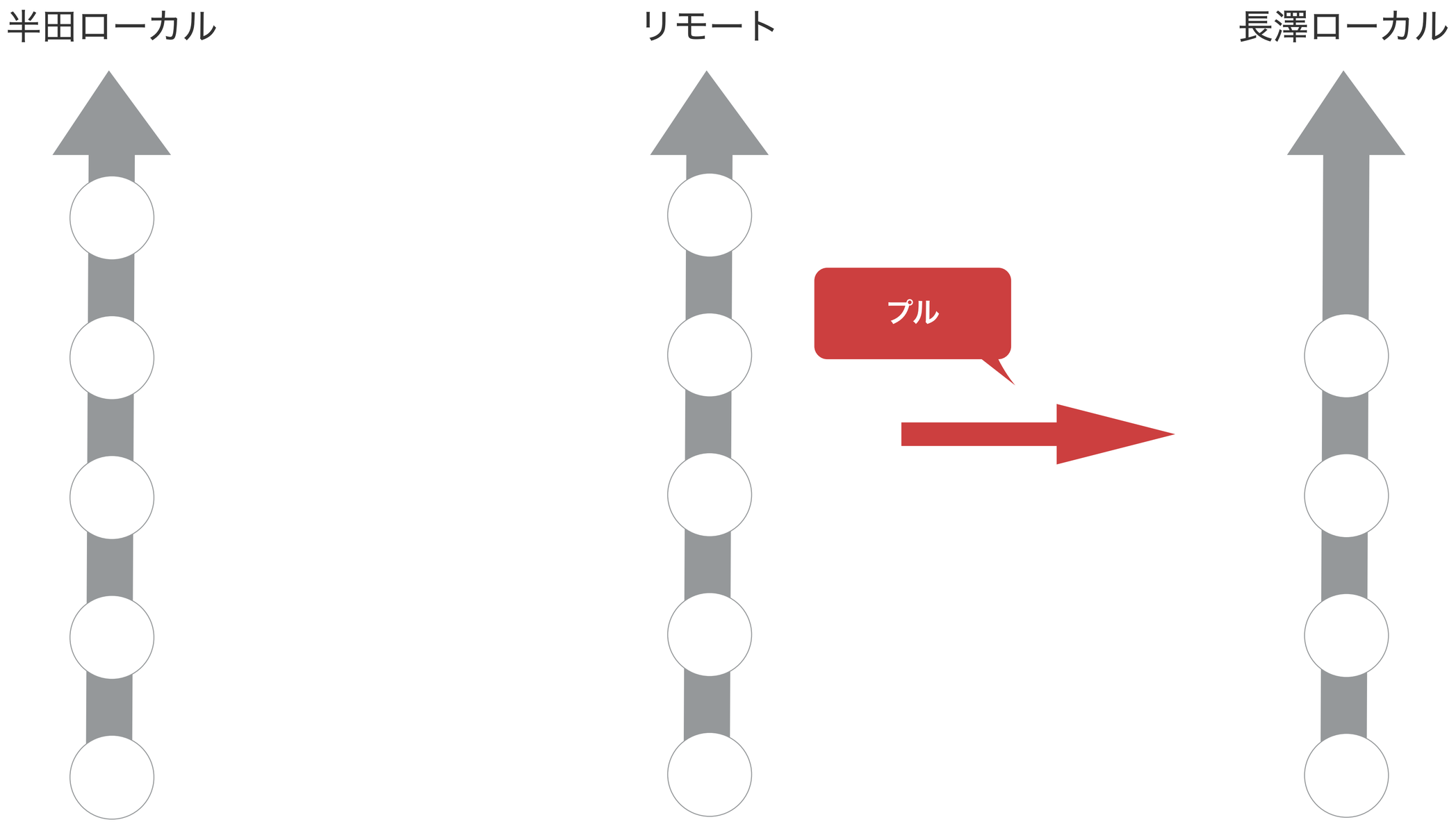
リモート



長澤ローカル

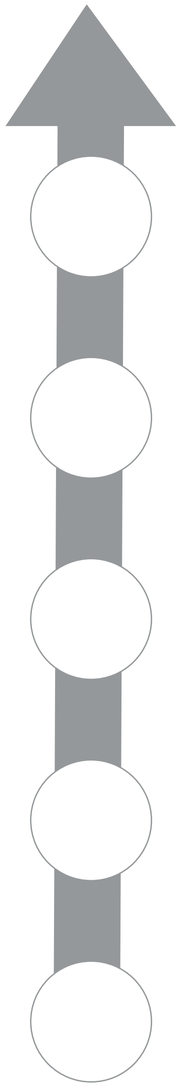


複数人での作業

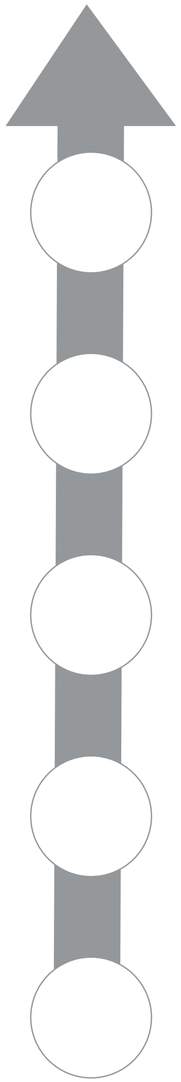


複数人での作業

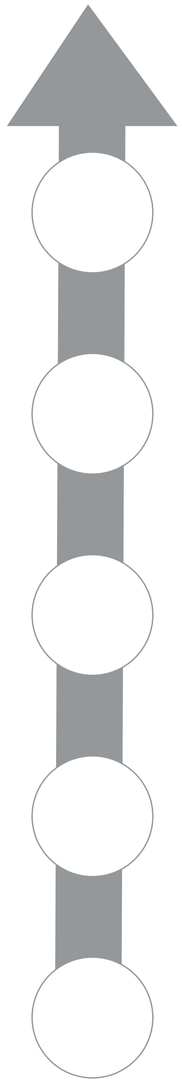
半田ローカル



リモート



長澤ローカル



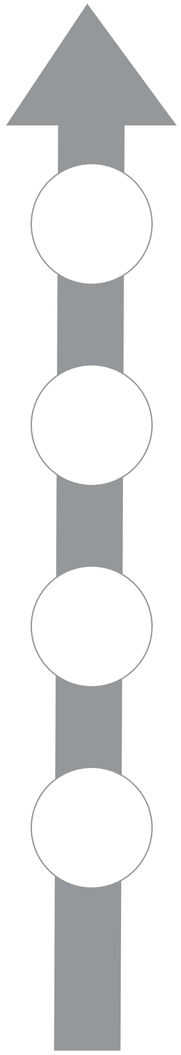
用語解説

ブランチ

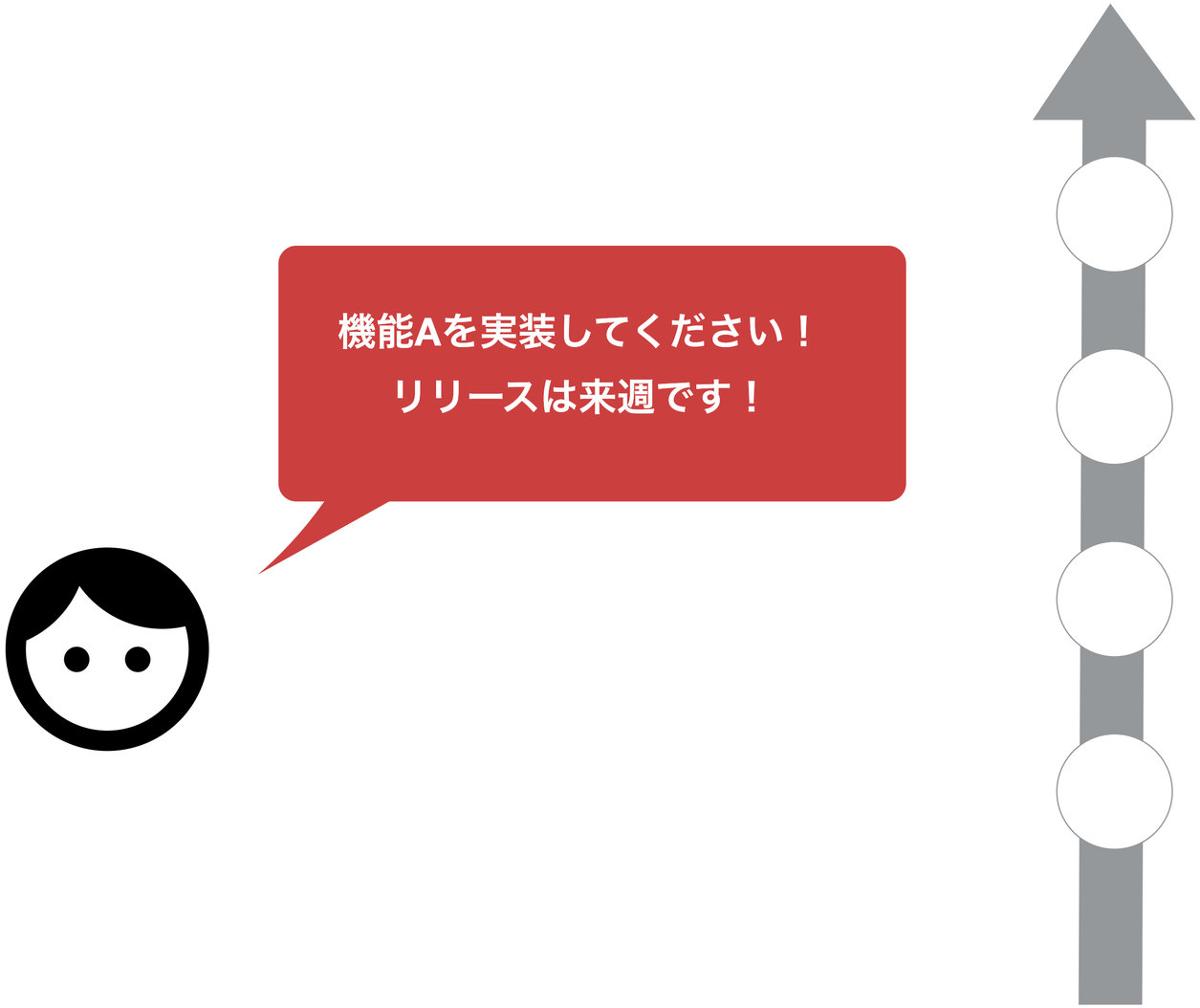
直訳すると枝。
作業を切り分けるためのもの。
百聞は一見にしかず。

ブランチ

まずはこれ。
実はmasterブランチっていいます。
（一本だけなのに枝とは）



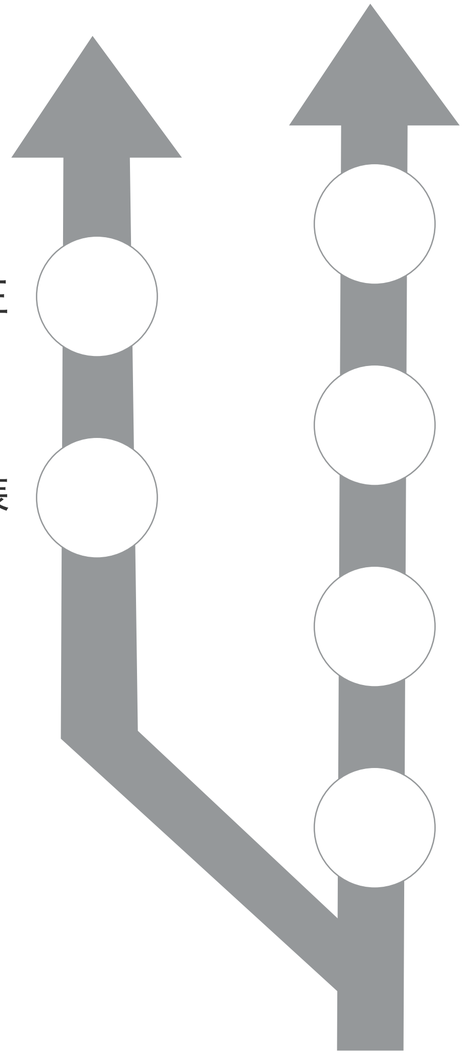
ブランチ



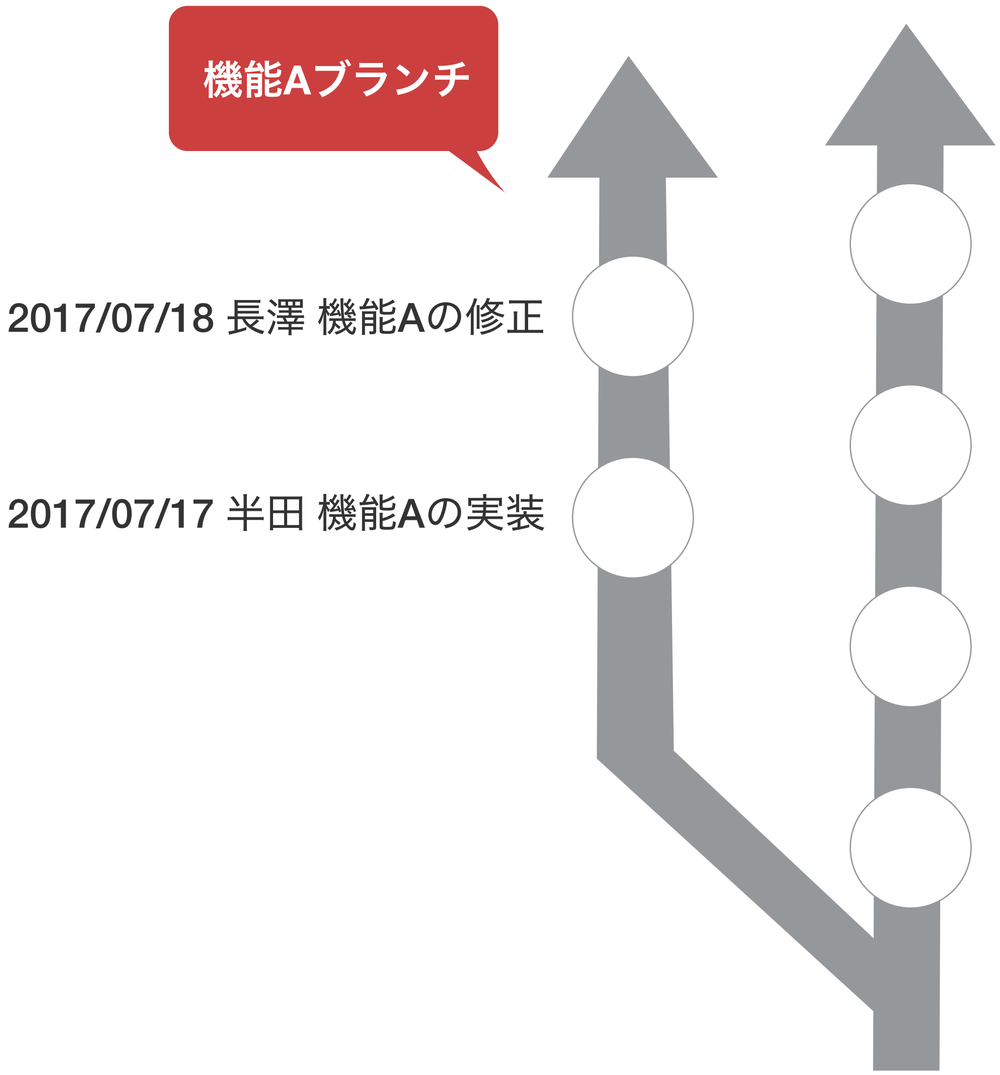
ブランチ

2017/07/18 長澤 機能Aの修正

2017/07/17 半田 機能Aの実装



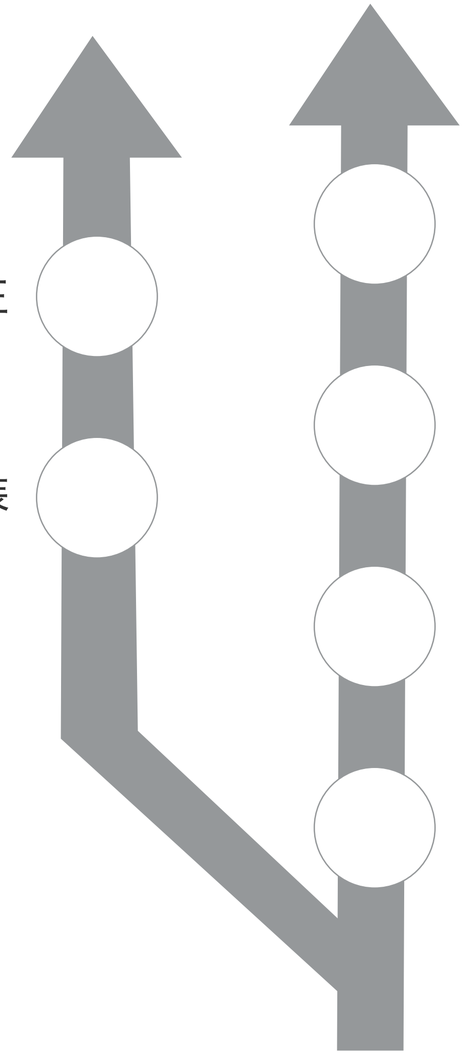
ブランチ



ブランチ

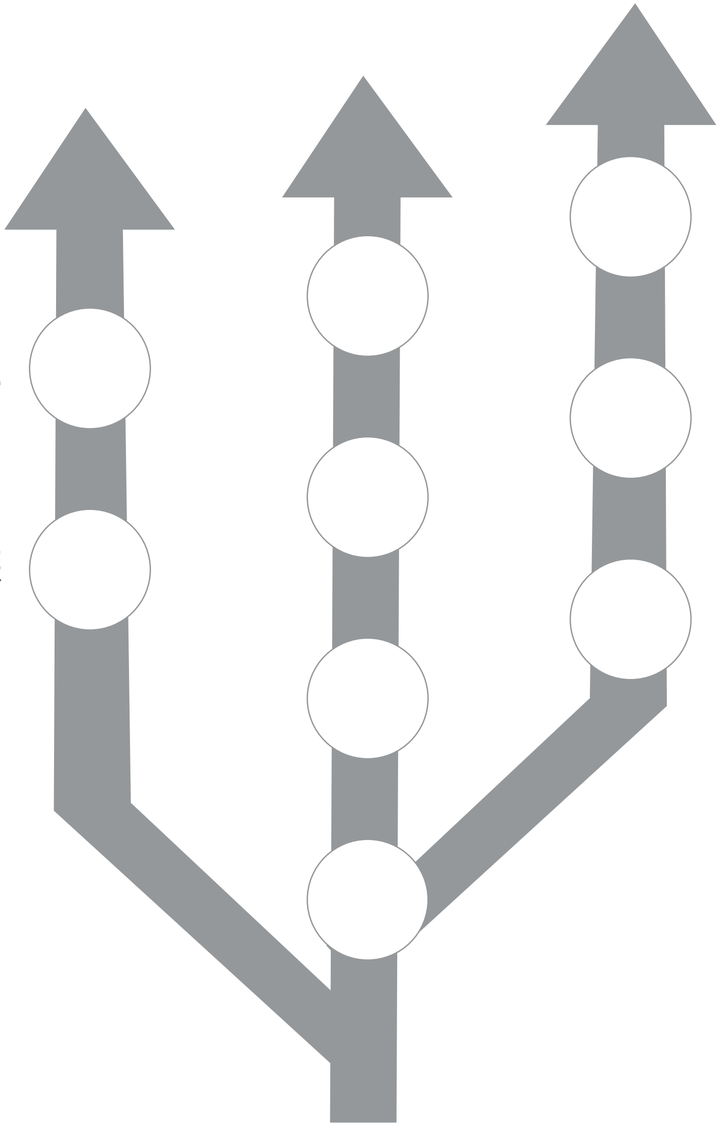
2017/07/18 長澤 機能Aの修正

2017/07/17 半田 機能Aの実装



機能Bを実装してください！
リリースは再来週です！

ブランチ



2017/07/18 長澤 機能Aの修正

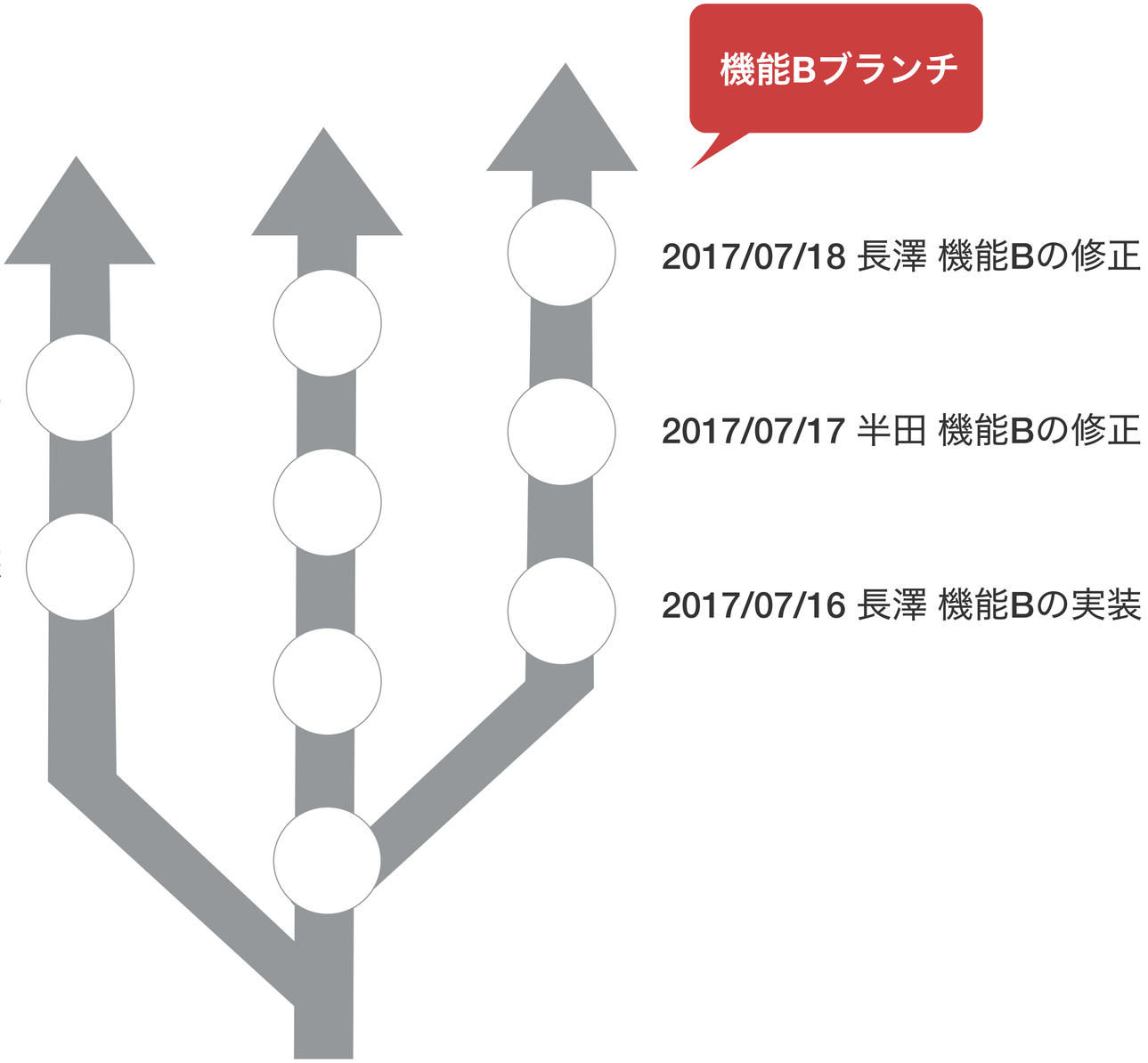
2017/07/17 半田 機能Aの実装

2017/07/18 長澤 機能Bの修正

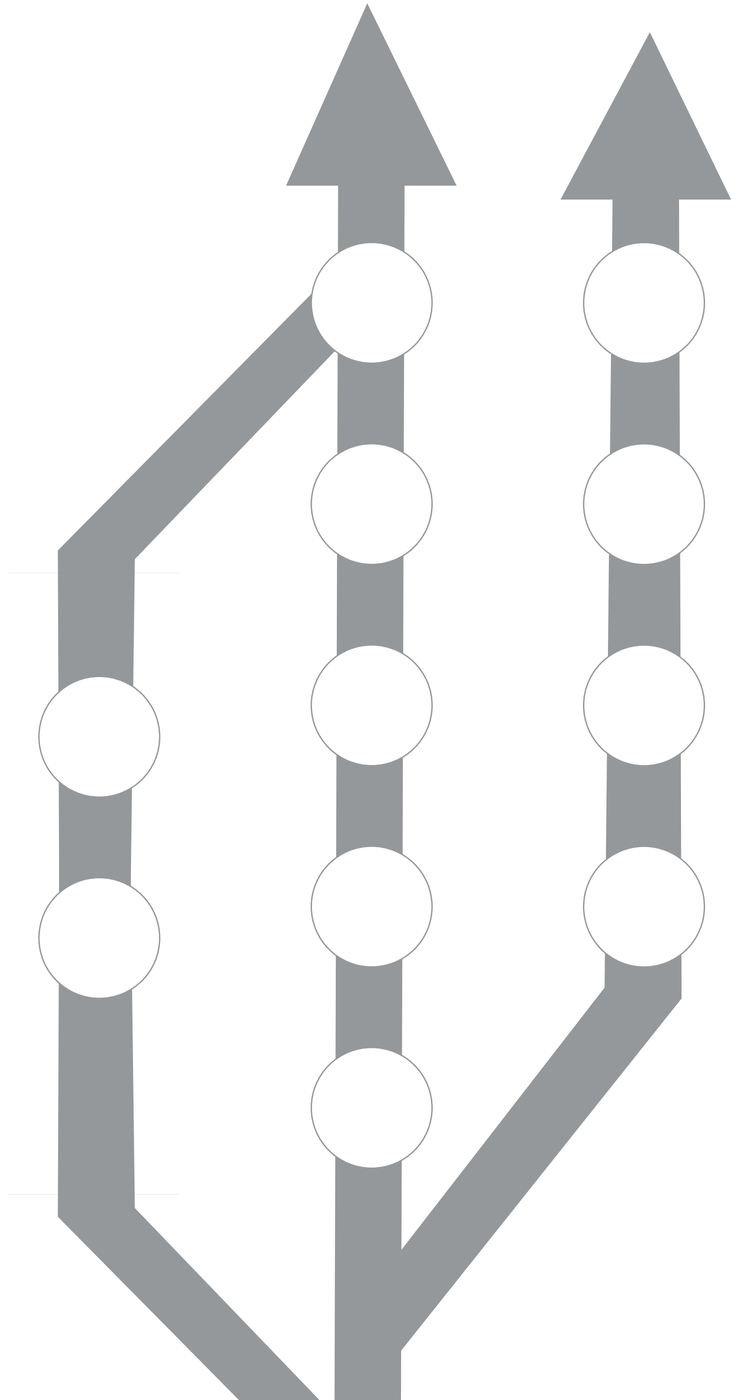
2017/07/17 半田 機能Bの修正

2017/07/16 長澤 機能Bの実装

ブランチ


機能Bブランチ
2017/07/18 長澤 機能Bの修正
2017/07/17 半田 機能Bの修正
2017/07/16 長澤 機能Bの実装
2017/07/18 長澤 機能Aの修正
2017/07/17 半田 機能Aの実装

機能Aリリース完了！



2017/07/18 長澤 機能Bの修正

2017/07/17 半田 機能Bの修正

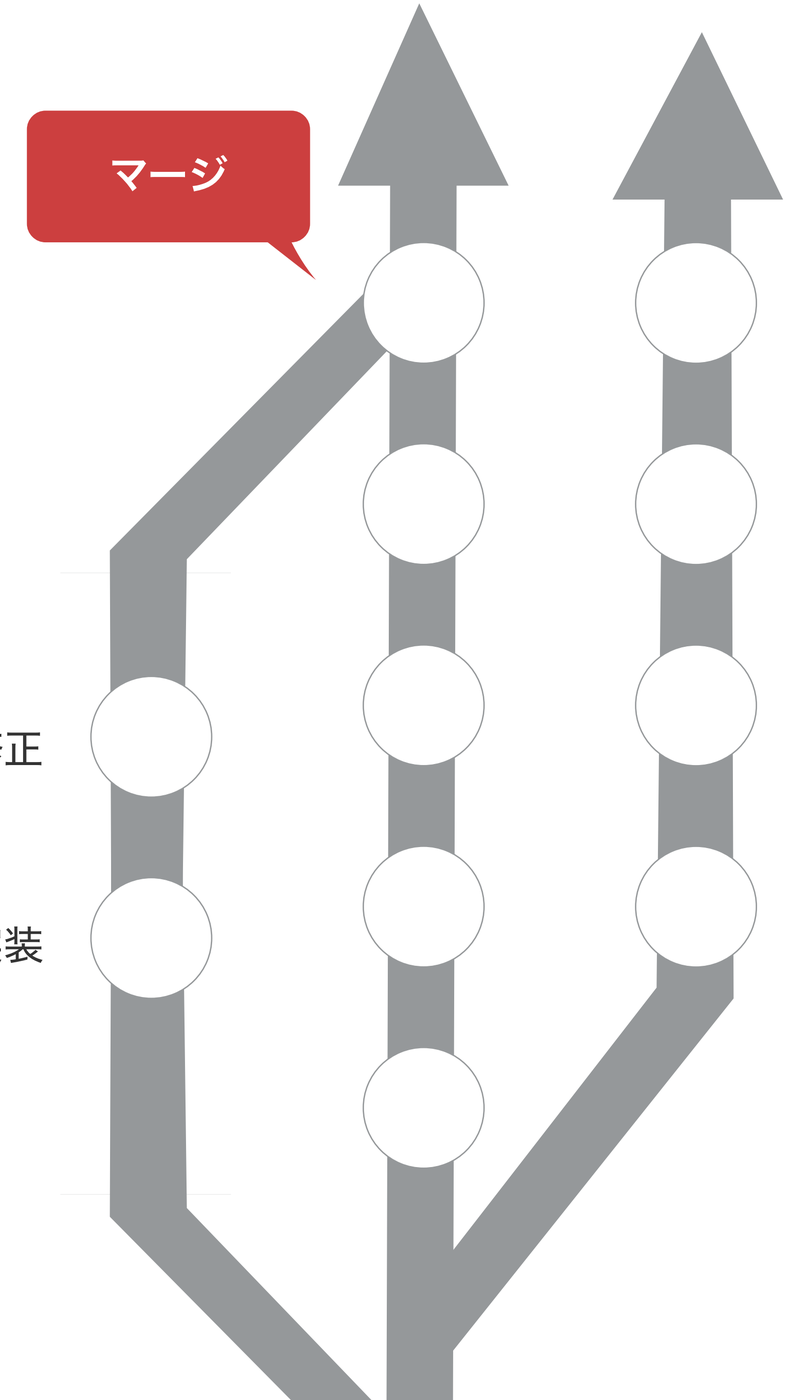
2017/07/16 長澤 機能Bの実装

2017/07/16 長澤 機能Bの実装

2017/07/18 長澤 機能Aの修正

2017/07/17 半田 機能Aの実装

機能Aリリース完了！



2017/07/18 長澤 機能Aの修正

2017/07/17 半田 機能Aの実装

2017/07/18 長澤 機能Bの修正

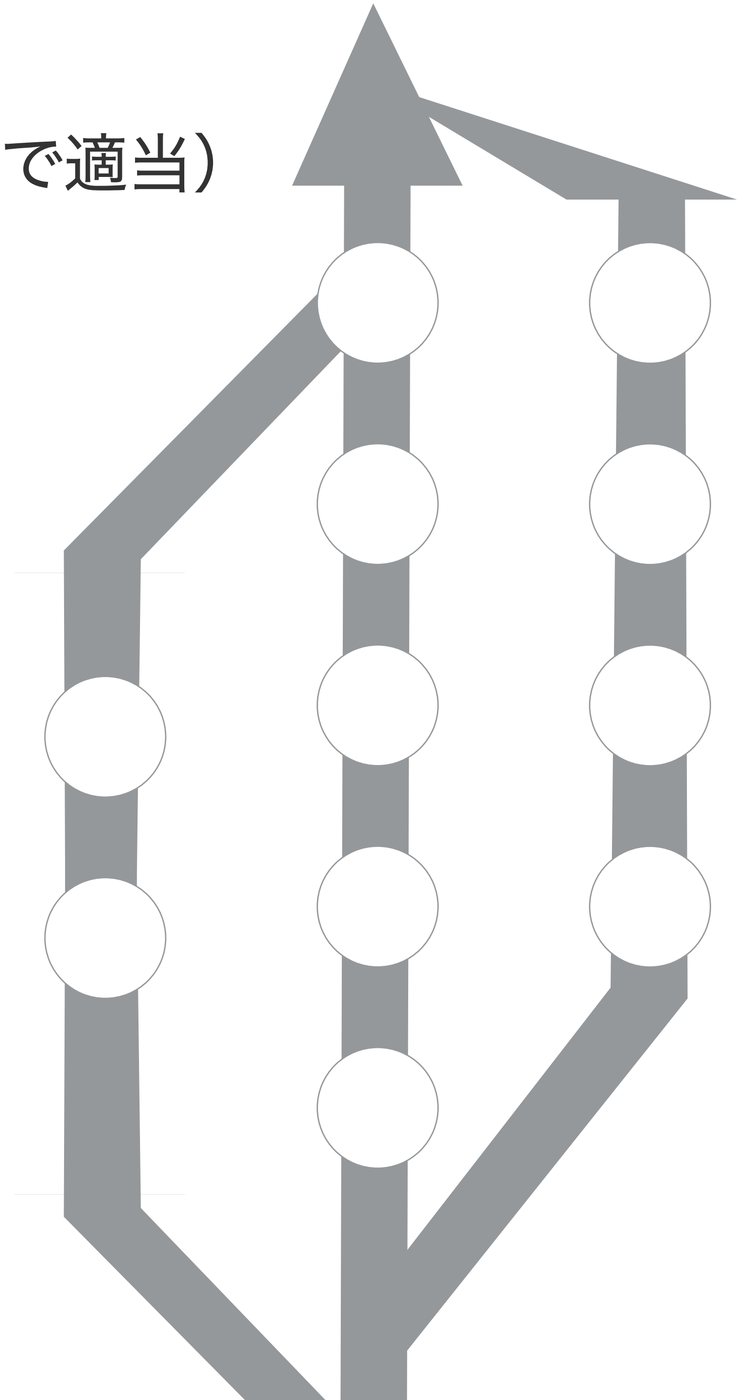
2017/07/17 半田 機能Bの修正

2017/07/16 長澤 機能Bの実装

2017/07/16 長澤 機能Bの実装

機能Bリリース完了！

（シェイプいじるのつらいので適当）



2017/07/18 長澤 機能Bの修正

2017/07/17 半田 機能Bの修正

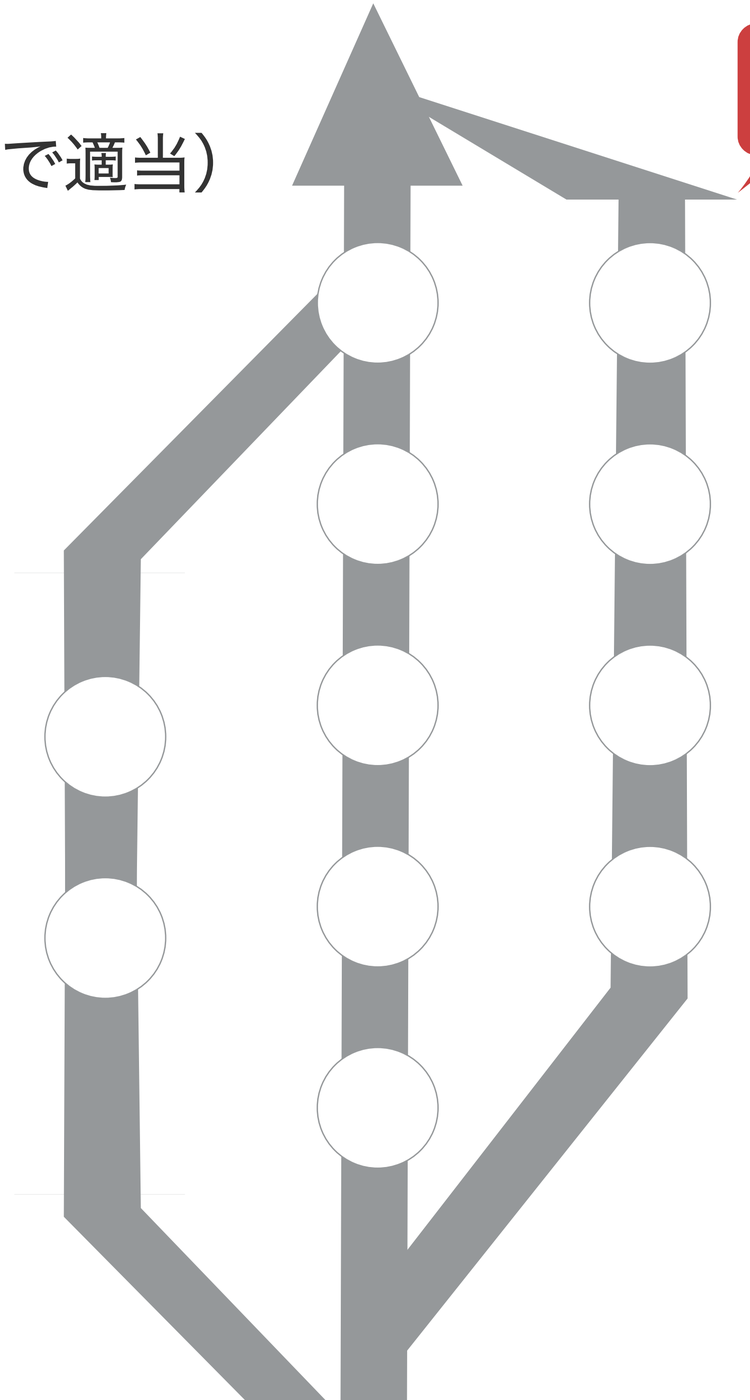
2017/07/16 長澤 機能Bの実装

2017/07/16 長澤 機能Bの実装

2017/07/18 長澤 機能Aの修正

2017/07/17 半田 機能Aの実装

機能Bリリース完了！
（シェイプいじるのつらいので適当）



2017/07/18 長澤 機能Bの修正

2017/07/17 半田 機能Bの修正

2017/07/16 長澤 機能Bの実装

2017/07/16 長澤 機能Bの実装

2017/07/18 長澤 機能Aの修正

2017/07/17 半田 機能Aの実装

用語解説

マージ

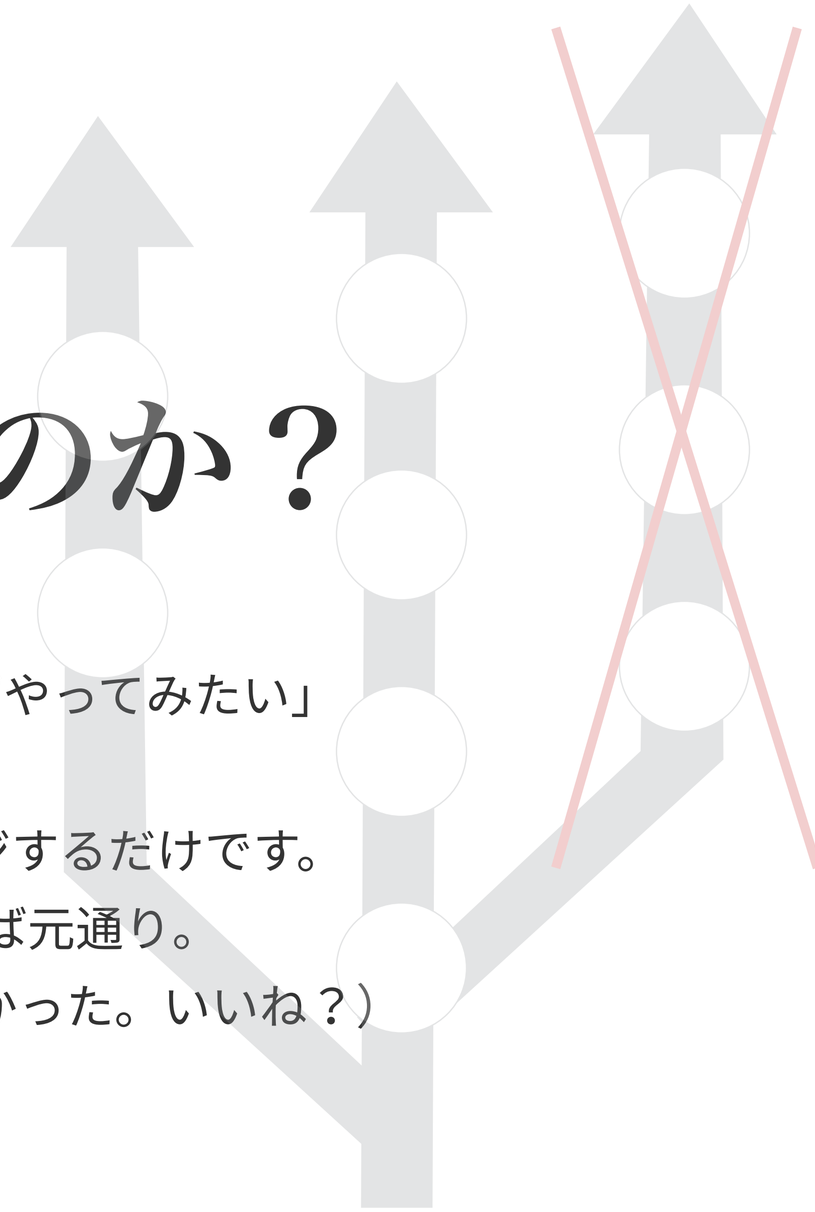
コミットの整合性を保ってくれるやつ。
問題がなければそのまま通るし、
「同じファイルの同じ場所を違う人がいじってるよ！」
というような事故がありそうな場合は、コンフリクトを教えてくれる。

ブランチは何がいいのか？

先ほどの例で言えば、機能Aブランチでは他に進行中のブランチのファイル
変更の影響を受けなくなるので、作業Aの開発にのみ集中することができます。
また、作業Aで変更したファイルも他のブランチに影響を及ぼしません。
つまり専用の開発スペースを作るようなイメージです。

用語解説

ブランチは何がいいのか？



「本当に実装するかどうかわからないけど、とりあえずやってみたい」
というときもブランチが活躍します。
良さそうだったら、ブランチをそのままmasterにマージするだけです。
やっぱりダメだったら、ブランチごと切り捨てれば元通り。
（機能Bブランチなんて存在しなかったし、君は何も見なかった。いいね？）

注意！

ブランチ&マージ

自動マージはあくまで、テキストベースのファイルのためのものです（というかGit自体）。
画像やOfficeファイル、デザインデータ等の中身が人間が読む文字列で表せない
ファイル（バイナリファイルといいます）は自動でマージ出来ません。
一見マージが上手くいったように見えても、どちらかの作業内容が消失してるので注意してください。
（コミットしている限りは過去の作業は記録されているので安心してください）

バイナリファイルの運用方法①

同じファイルを同時にいじらない

基本的に「同じファイルを同じタイミングで編集しない」が原則。
そこは正直コミュニケーションで解決するのが1番いいかと思います。
同じタイミングでも、別ファイルであれば大丈夫です。

バイナリファイルの運用方法②

マージ後に手動でマージ

コンフリクトが起きた場合は、ひとまずGit上では解決させてコミットするしかありません
（実際はデータのマージができていない）。その後、
①いったん取り出したい作業内容がある過去のコミットに移動（チェックアウト）
②Git外のディレクトリにファイルをコピー
③最新のコミットに戻り、マージによって消失してしまった作業内容を手動でコピペして反映する
かなと思います。めんどくさい。

バイナリファイルの運用方法③

ブランチどうする？

バイナリファイルでは「複数人が同時に作業しない」が前提になるので、
あまりブランチは活用しなくても良いかなと思います（むしろなるべく使わない方が良いかも）。
少なくとも、マージを期待してはいけません。
（バイナリだとfast-fowardですらダメな気がする）

Gitにバイナリファイルは向いてない？

そんなこともない。
使用しているサービス・ツールによっては、画像やPSDの変化をきちんと表示してくれます。
また、デザインバージョン管理サービスが幾つかローンチしていますが、
多くはGitをベースに独自拡張しているものです。

GitHubのPSD差分表示

青に変更した

master

Browse files

247handa committed 7 minutes ago　　1 parent d072f8d　　commit 75e5d6fa0a1e1b443bba11ed2b40f3971599da62

Showing 1 changed file with 0 additions and 0 deletions.

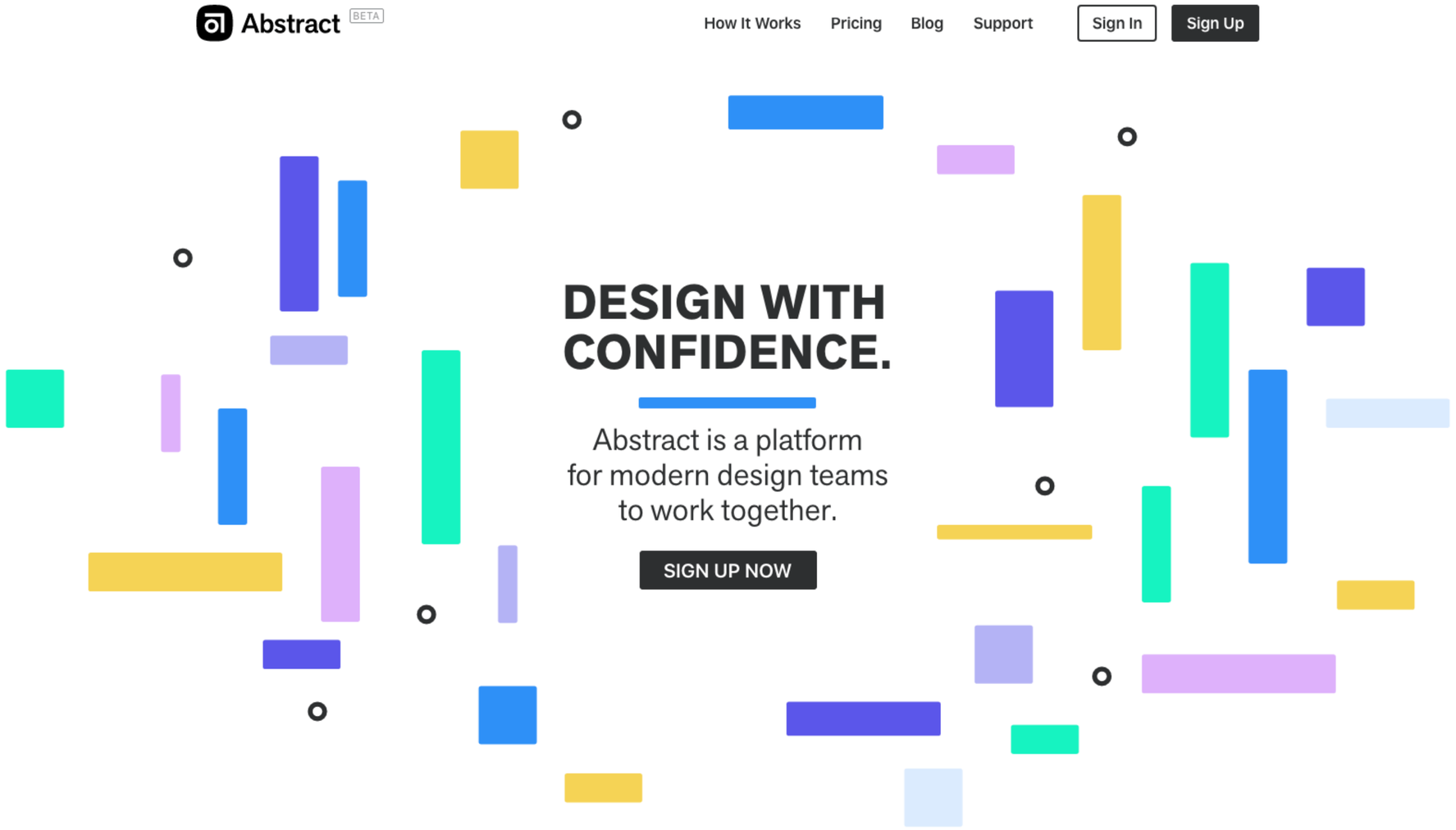
Unified　Split

BIN ■■■■■ precss.psd　　View

Deleted

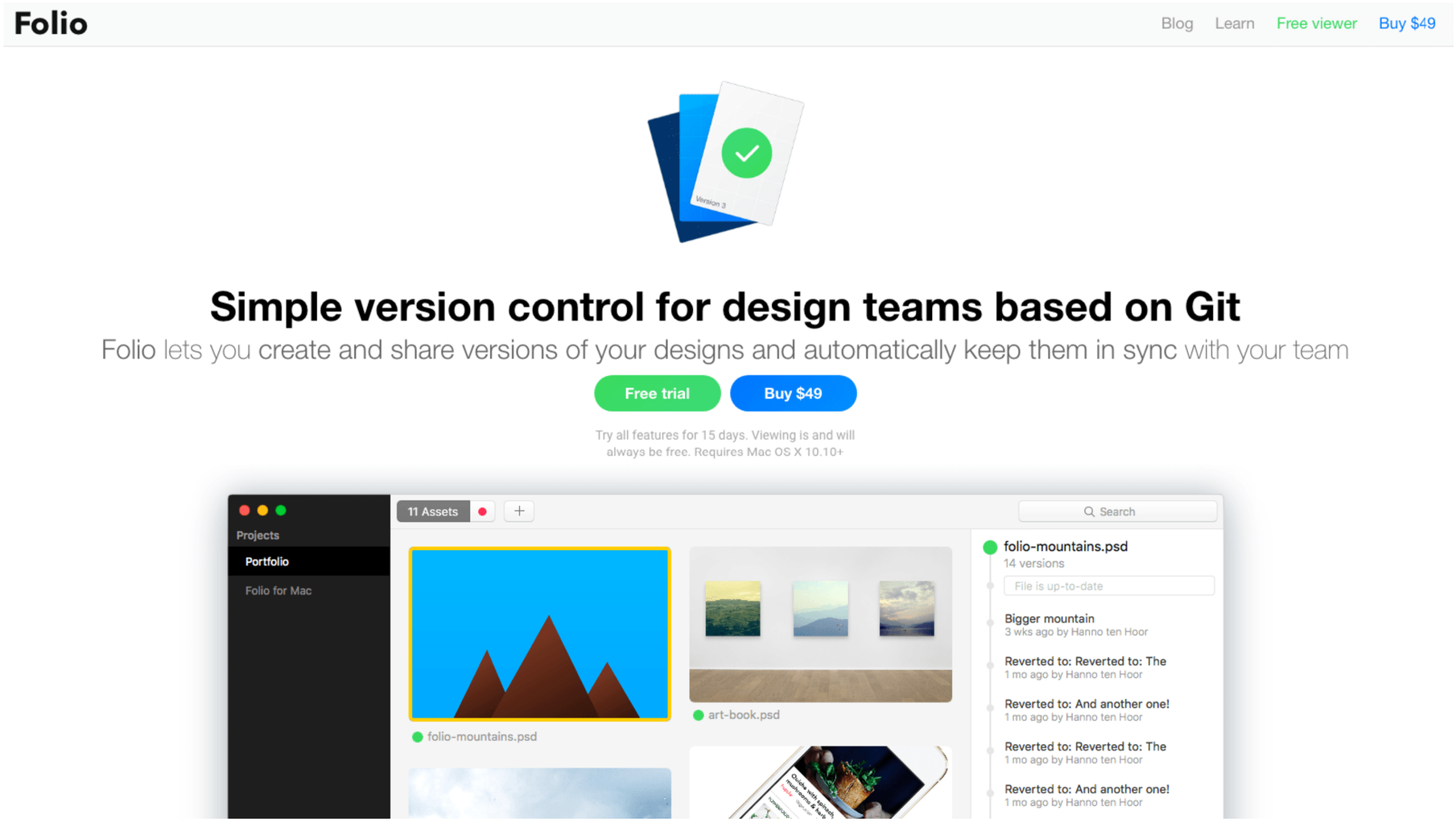
Added

中身はGitです



https://www.goabstract.com/

中身はGitです



http://folioformac.com/

でもGit重い

サイズの大きいバイナリファイルを管理するとどうしても重くなってしまうのは事実です。
それを解決するために、Git LFSという拡張機能が開発されています。
詳細はこちら↓

「大容量ファイルもGitで管理。 Git LFSの使い方」
https://goo.gl/moVrq3

Gitを始めるには

この辺りは幾らでも情報があるので、そちらにお任せします。
とりあえず「サルでもわかるGit入門」をご覧ください。
http://www.backlog.jp/git-guide/

Gitに慣れる

ローカルリポジトリだけで動かすことができるのもGitの特徴です。
まずは自分のPC内だけでも良いので、Gitを取り入れてみて慣れることをおすすめします。
機能はたくさんありますが、普段頻繁に使うのはせいぜい5, 6 個です。
→続く

Gitに慣れる

Gitといえど、ローカルだけだとストレージが壊れたら終わりなのでバックアップは取ってください。
やはりリモートリポジトリも作っておくのが1番です。そこで始めやすいのがBitbucketです。
無料で幾つでもプライベートリポジトリ（他の人には見えないリポジトリ）を作ることができます。

ちなみにBitbucketはSouceTreeを作っている会社のサービスなので、SourceTreeとも親和性が高いです。
https://bitbucket.org/

コミットの粒度を気にする

「より良い運用」のために気をつけることは幾つかありますが、まずはコミットの粒度を気にしてください。
例えば機能Aと機能Bの実装を1つのコミットにして機能Cに不具合が発生した場合、
AとBどちらが原因なのか分かりません。

後から遡ったときにチェックしやすいように、機能ごとやページごと、
区切りの良いところでコミットを分けるようにします。

コミットメッセージを分かりやすく

コミットメッセージを分かりやすく、簡潔にすることも重要です。
「変更した」とだけ書いた場合、コミットの中身を見に行かないと何を変更したのか分かりません。
バランスが難しいですが、ポイントはとにかく「後から見たときに分かりやすいか」です。

例えば弊社では、情報不足を補うためにタスク管理ツールの
チケットID、コメントIDを必ず紐付けるようにしています。
例：[チケットID#コメントID] トップページに◯◯を実装

まとめ

エンジニアはもはや必須ですが、
Gitの恩恵を受けられるのは
エンジニアだけではありません。

実際にこのスライドも、弊社（パンセ）でディレクター・デザイナーもGitを使ってドキュメントや
デザインデータのファイル管理をしていくことになり、その説明用として作りました。

デザイナー向けのバージョン管理
サービスも、根幹にあるのはGitです。

デザインデータに限らずこの流れは
拡大していくと思うので、
仕組みを知ってると有利かと。

Git、便利です。