GITについて

計算機科学コース　中澤篤志
Revised by 五十嵐淳

参考：計算機科学実験及び演習のための Git(Yusuke Miyazakiさん)

バージョン管理システムとは

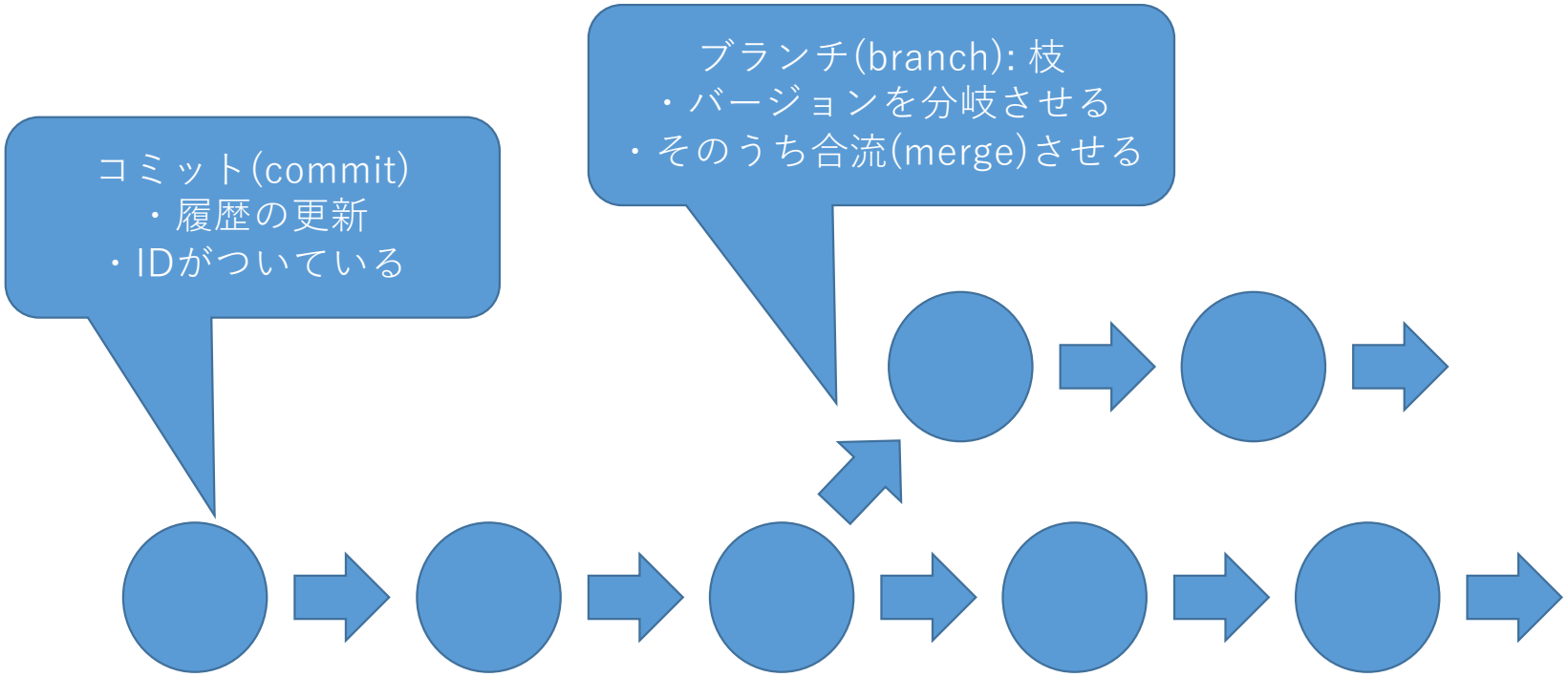
- **バージョン管理システム**とは、コンピュータ上で作成、編集されるファイルの変更履歴を管理するためのシステム。特にソフトウェア開発においてソースコードの管理に用いられる。
- バージョン管理システムの最も基本的な機能は、ファイルの作成日時、変更日時、変更点などの履歴を保管することである。これにより、何度も変更を加えた ファイルであっても、過去の状態や変更内容を確認したり、変更前の状態を復元することが容易になる。

要は

- プログラムのファイルを管理できる
- プログラム変更点などを管理できる
  - 誰が変更したか？を記録
  - 誤った変更を元に戻せる
  - 何を変更したかを確認できる
- Gitは現在最もよく用いられているバージョン管理システムの１つ

Gitの管理イメージ

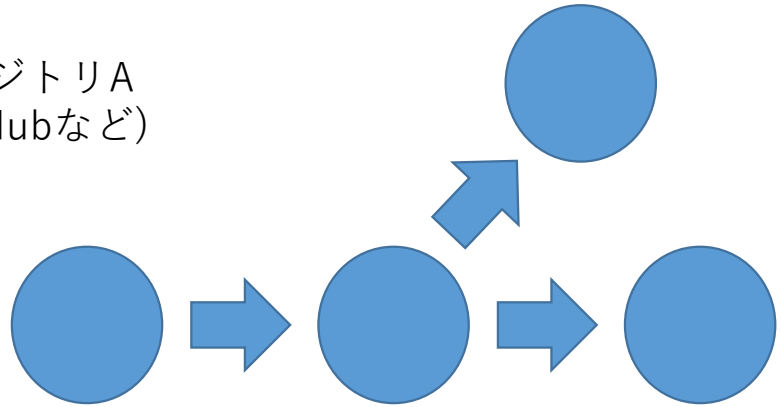
管理データ全体を
レポジトリ（repository）
という



世代管理

Gitの*分散*管理イメージ

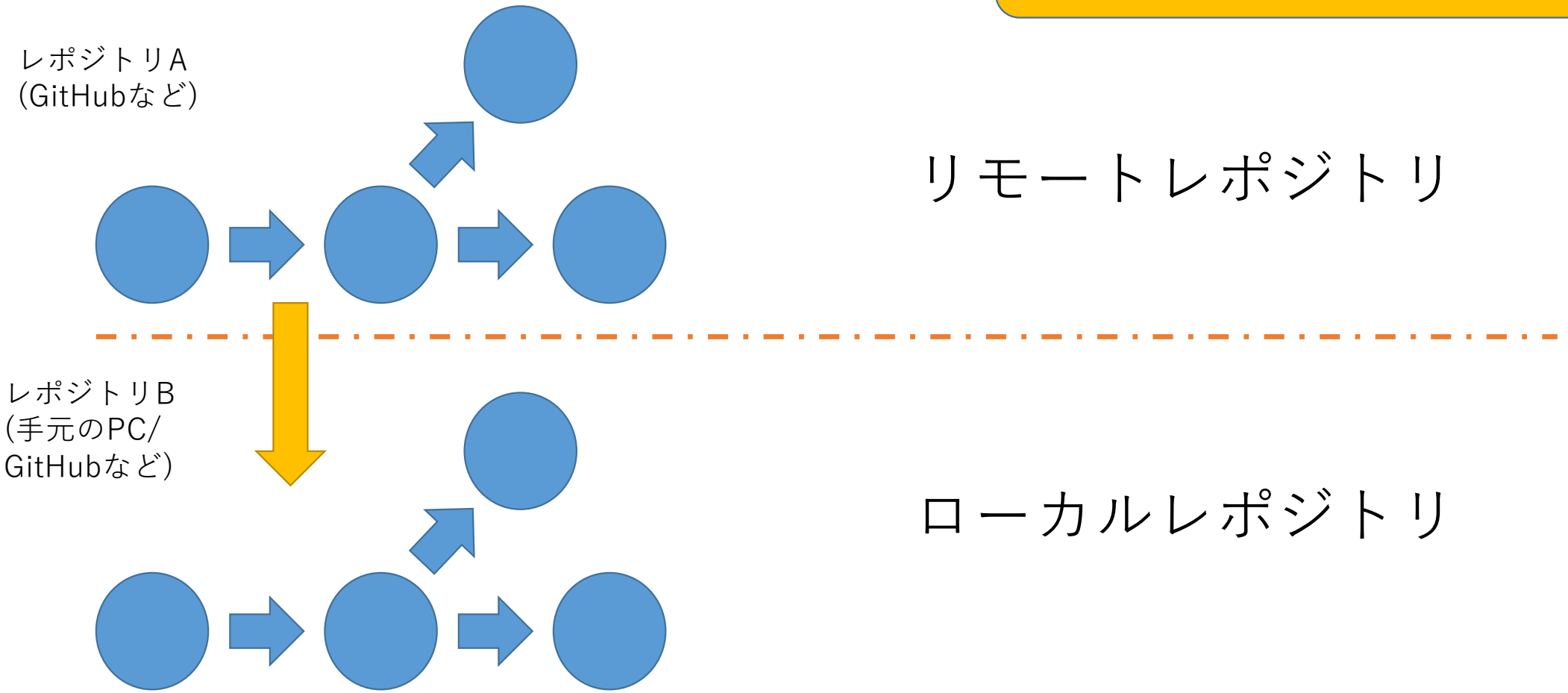
レポジトリA
(GitHubなど)



レポジトリB
(手元のPC)

Gitの*分散*管理イメージ

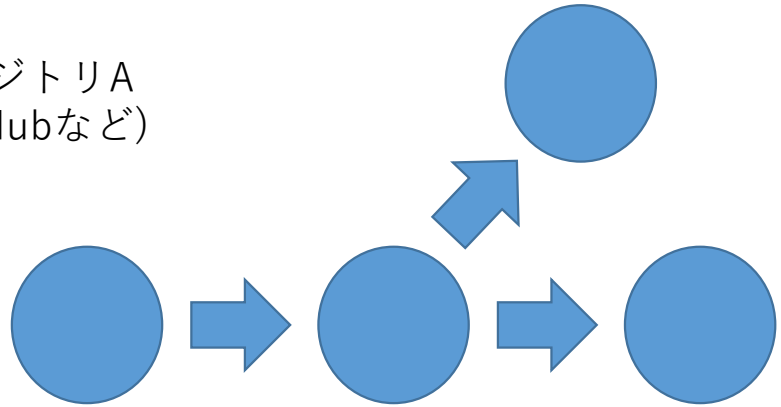
1. A の複製を手元に作る (clone)



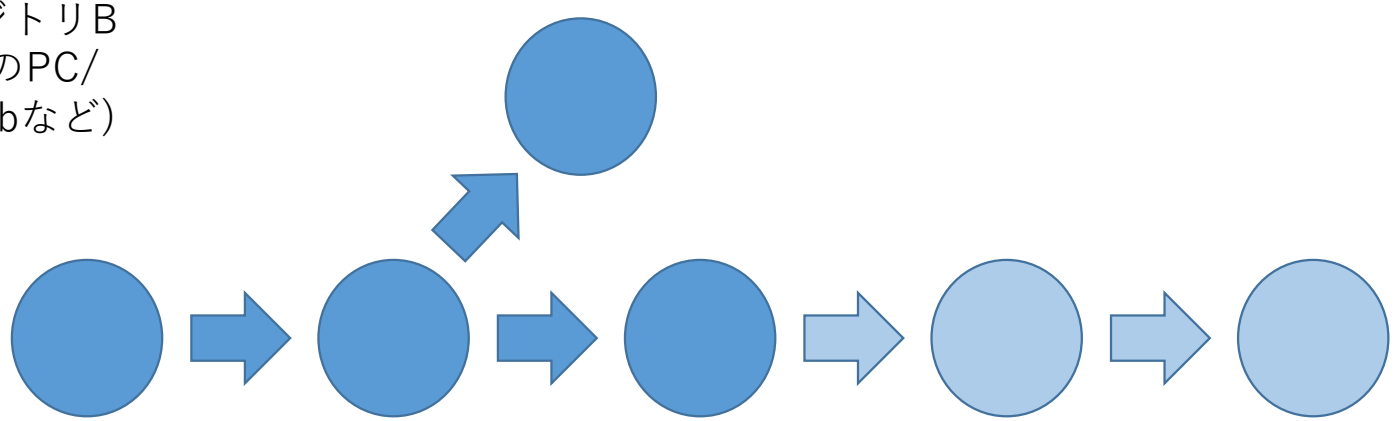
Gitの*分散*管理イメージ

2. Bで開発を進める
(commit の作成)

レポジトリA
(GitHubなど)

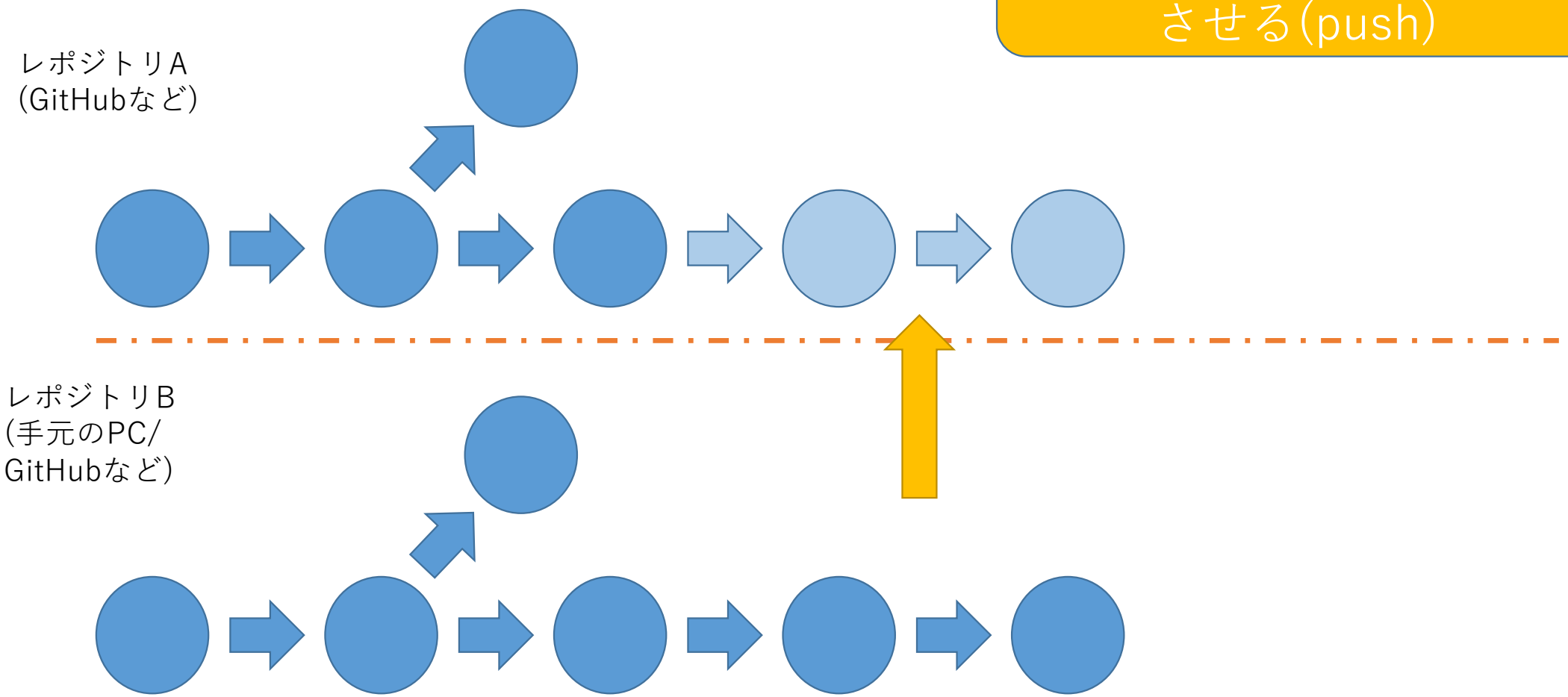


レポジトリB
(手元のPC/
GitHubなど)



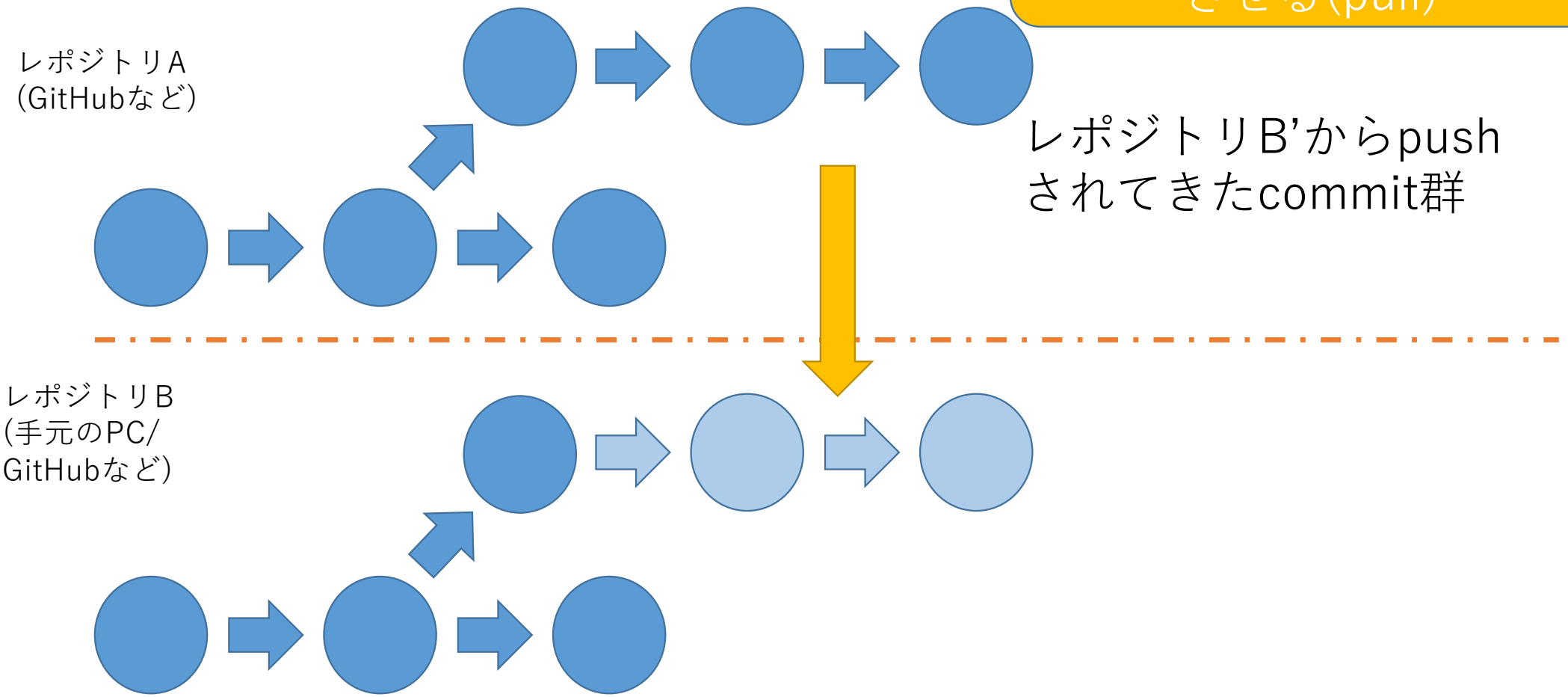
Gitの*分散*管理イメージ

3. ローカルブランチのcommitをリモートに反映させる(push)



Gitの*分散*管理イメージ

3’. リモートブランチのcommitをローカルに反映させる(pull)



注意!

- ここからの説明は、macOS、Windows/WSL2/Ubuntu、Ubuntu Linux での(いわゆるターミナル画面での)操作を仮定しています
- 以下のことについては説明しません
  - Windows と WSL2 環境間でのファイル共有
  - プログラムファイルの編集方法
    - VSCode を Windows にインストールして拡張機能を使うと、編集は VSCode で、コンパイルや Git は WSL2 側で実行できるらしいです。
      - https://learn.microsoft.com/ja-jp/windows/wsl/tutorials/wsl-vscode
    - IntelliJ にもおそらく類似機能があります
      - https://pleiades.io/help/idea/how-to-use-wsl-development-environment-in-product.html#open-a-project-in-wsl かな?

コマンドライン操作の基本のき

コマンドを実行する時の「現在位置(カレント・ディレクトリ)」が大事

- **pwd** …現在位置を確認するコマンド
- **ls** …現在位置にあるファイル一覧の表示
- **cd** 〈ディレクトリ名〉 … 現在位置にある〈ディレクトリ名〉というディレクトリに移動(現在位置を変更)する
  - **cd ..** ひとつ上のディレクトリ(親ディレクトリ)に移動
  - **cd** 自分のホームディレクトリに戻る
- **cat** 〈ファイル名〉 … 現在位置にある〈ファイル名〉というファイルの内容を表示する(ファイルの内容によっては端末画面が死にます(ぉぃ))

リポジトリの作成

バージョン管理を始めるため、履歴を格納するGitリポジトリを作成

1. プロジェクトのルートディレクトリで

`git init`

を実行する→空の(更新履歴がない)レポジトリができる

- 一度だけ行えばよい
- 該当するディレクトリ内に .git という隠しフォルダができる

今回はこっち

1’. リモートレポジトリから clone する

`git clone` 〈リモートレポのURL〉〈ディレクトリ名〉

→現在位置に〈ディレクトリ名〉というディレクトリができてその中にリモートレポの変更履歴と現在のファイル群がコピーされる

GitHub からの clone の方法(SSH編)

1. 鍵ペアの作成とssh-agentへの登録
   (https://docs.github.com/ja/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent)
2. 生成した公開鍵をGitHub側の自分のアカウントへ登録
   (https://docs.github.com/ja/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account)
3. レポジトリをcloneするための URL を確認
   3-1. レポジトリをブラウザで開く
   3-2.〜3.4 右記参照
   3-5. git clone の実行

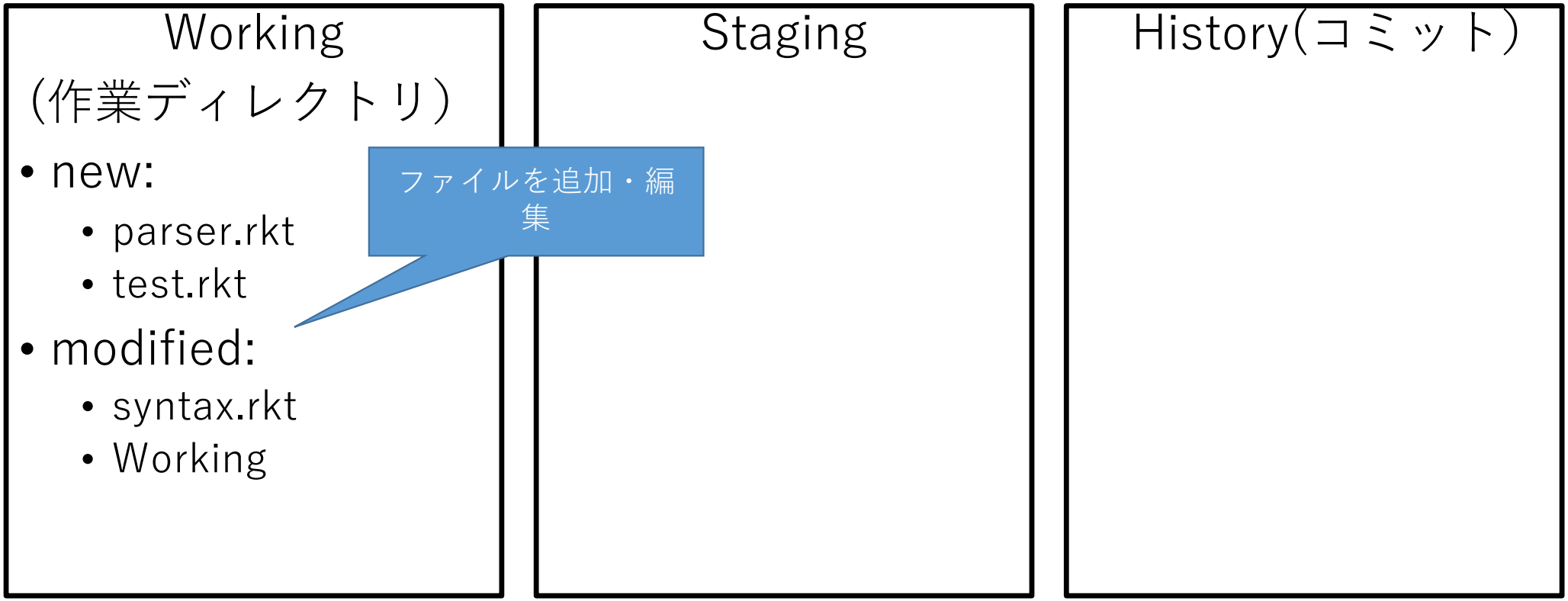
状態の確認

- 現在の Git リポジトリの状態を確認するには、レポジトリのディレクトリに移動して

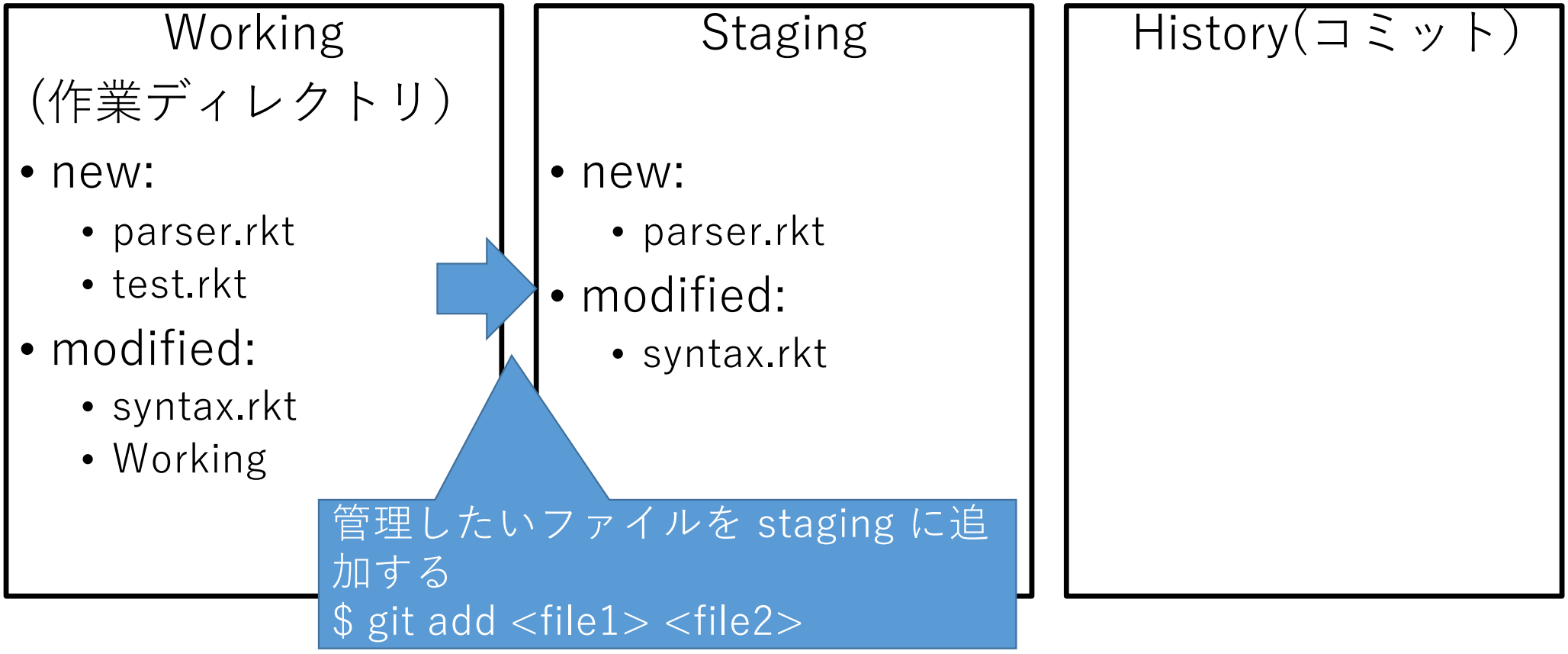
```
git status
```

- Working Dir. で追加・変更された内容やStaging に追加された内容が表示される

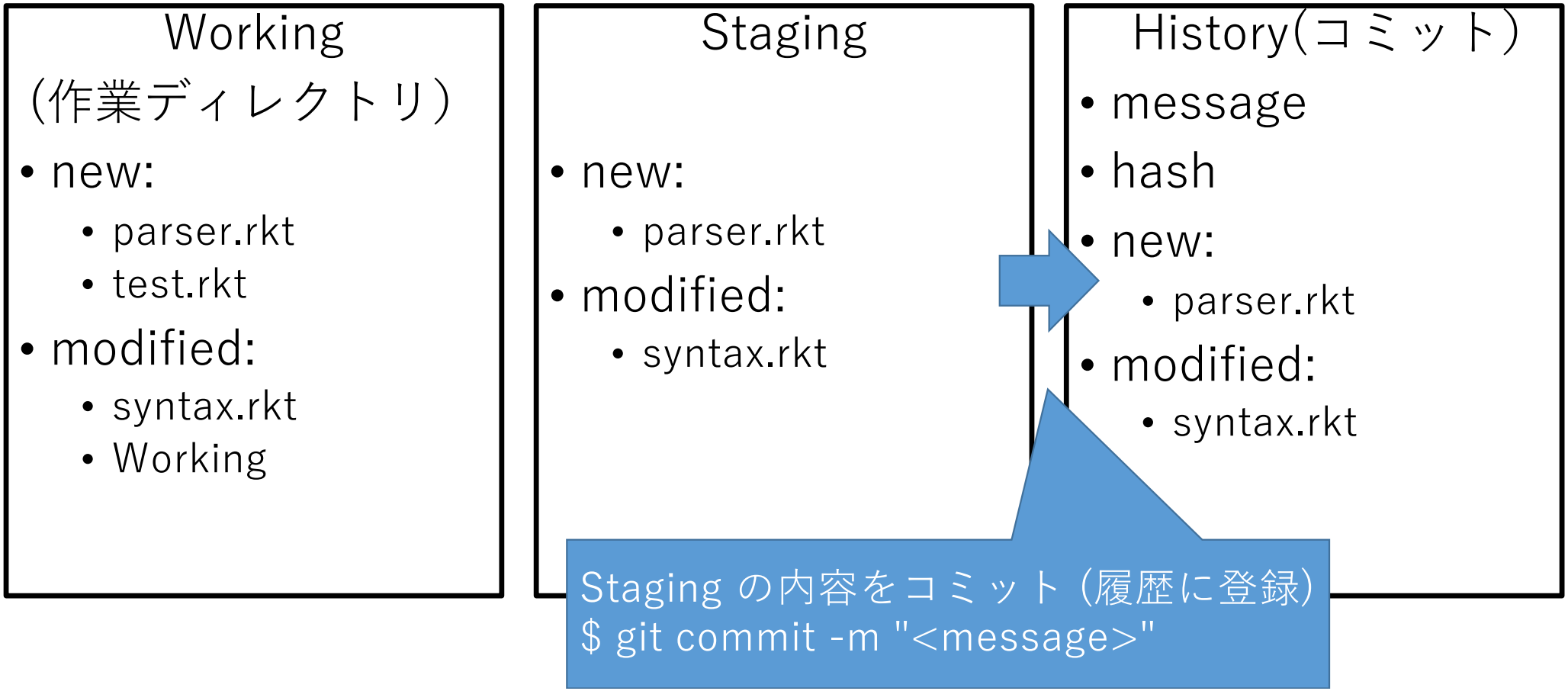
管理の流れ（１）


Working
(作業ディレクトリ)
• new:
• parser.rkt
• test.rkt
• modified:
• syntax.rkt
• Working
ファイルを追加・編集
Staging
History(コミット)

管理の流れ（２）


Working
（作業ディレクトリ）
• new:
• parser.rkt
• test.rkt
• modified:
• syntax.rkt
• Working
Staging
• new:
• parser.rkt
• modified:
• syntax.rkt
History(コミット)
管理したいファイルを staging に追加する
$ git add <file1> <file2>

管理の流れ（３）

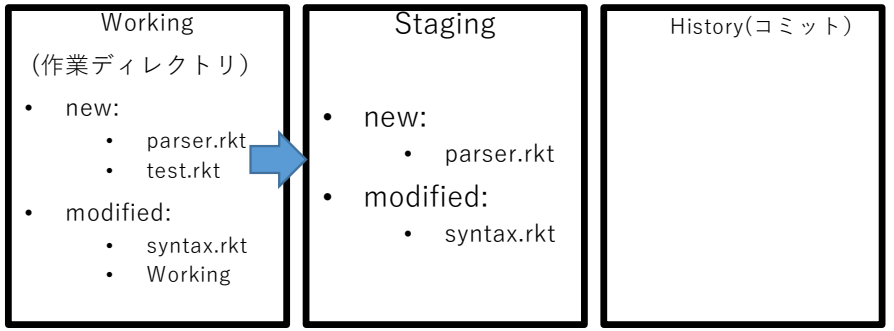

Working
（作業ディレクトリ）
• new:
• parser.rkt
• test.rkt
• modified:
• syntax.rkt
• Working
Staging
• new:
• parser.rkt
• modified:
• syntax.rkt
History(コミット)
• message
• hash
• new:
• parser.rkt
• modified:
• syntax.rkt
Staging の内容をコミット（履歴に登録）
$ git commit -m "<message>"

管理（１）

- 機能の追加やバグの修正などきりの良い時点でコミットし、状態を保存しておく

- 新しくバージョン管理下に置くファイルや変更したファイルを Staging に追加する

  - **`git add <file1> <file2>` …**

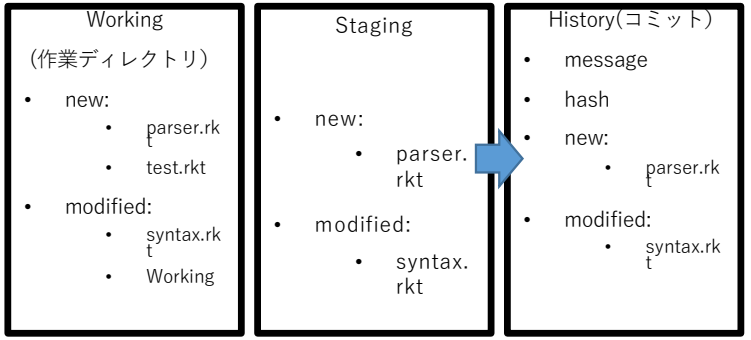


管理(2)

- Staging の内容をコミットして履歴に登録する

`git commit -m "<commit message>"`

- コミット時にはメッセージを付加する
- 変更内容を書いておくと後で分かりやすい
  - 例1: 課題1-1 完了
  - 例2: 課題1-5で〜〜〜になるバグを修正



管理(3)

- ローカルのコミットをリモートに “push” する
- リモートレポジトリ名の確認
  - **`git remote -v`**
    → **`origin <cloneした時のURL>`**　みたいなのが2行表示されるはず
- 現在のブランチ名確認
  - **`git status`**
    → 1行目に **`On branch main`** などと表示される
- Push!
  - **`git push origin main`**

履歴の閲覧1

- 履歴を閲覧する

```
git log
```

- コミットの一覧がハッシュ・メッセージ・日時などとともに表示される

- それぞれのコミットはハッシュ (SHA-1) で一意に特定できる
  - 例: 5a00b5712a039bfea1e8055206ab697e3081247d

- オプションを付加するともっと色々見れる

履歴の閲覧2

- 特定のコミットの変更内容を見る

```
git show <commit>
```

最後(最新)のコミットまで戻る

- 変更に問題があった場合、最後のコミットの状態に戻す

```
git restore <file or dir>
```

(ただし、commit していない変更があるときは戻れないのでキリがよい時にやる)