

講義資料配布ページ(各自参照)

<http://bit.ly/kosen02>

# プログラミング応用第2回 バージョン管理システムの基礎(1)

講義資料配布ページ(各自参照)

<http://bit.ly/kosen02>

4月21日

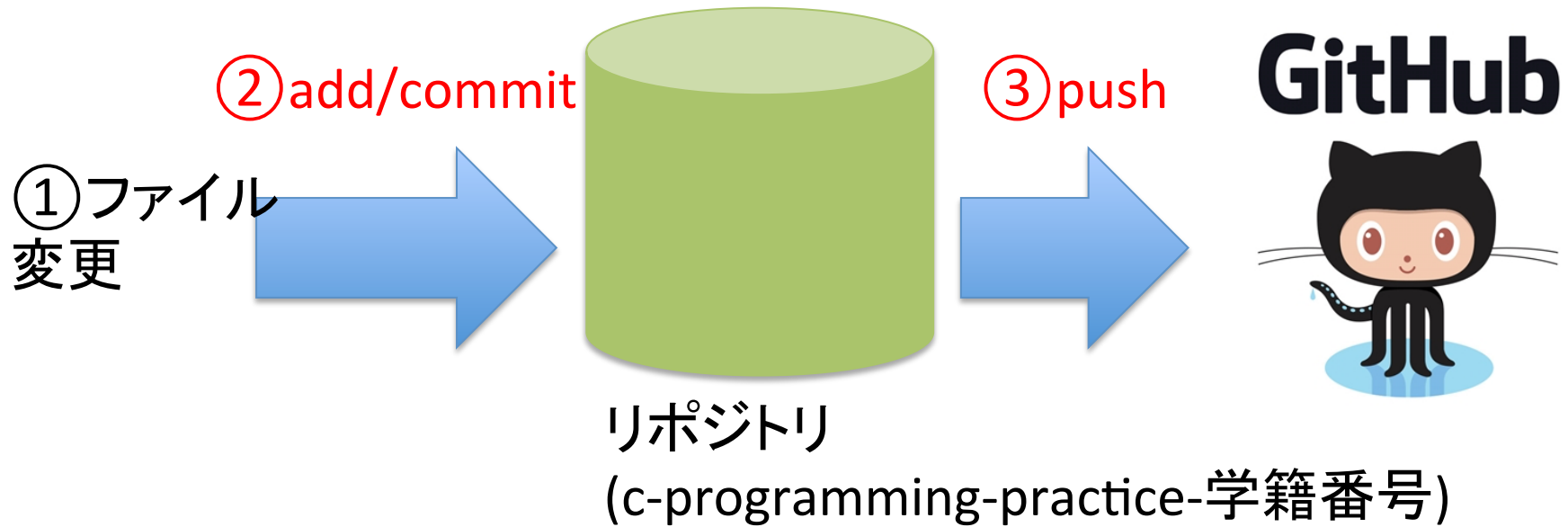
# 本日の講義

1. ブランチの基礎
2. バージョン管理システムを用いたソースコードの配布
3. バージョン管理システムを用いた共同開発
4. Git以外のバージョン管理システム  
(分散型/集中型)
5. ソースコードと著作権

# 本日の講義

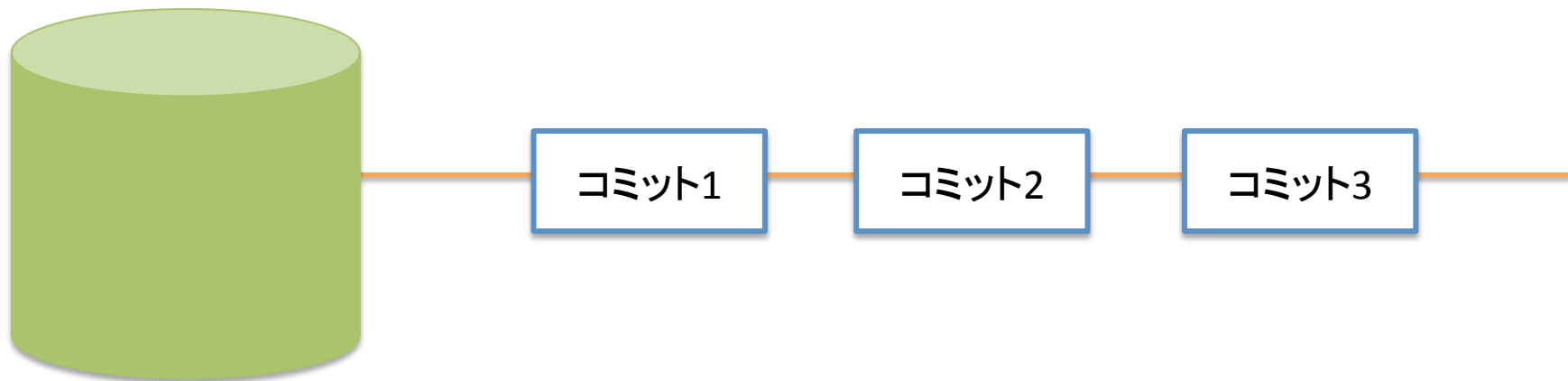
1. ブランチの基礎
2. バージョン管理システムを用いたソースコードの配布
3. バージョン管理システムを用いた共同開発
4. Git以外のバージョン管理システム  
(分散型/集中型)

# 復習：Gitをひとりで使う場合



# 復習:コミットによる変更履歴記録

- “リポジトリ”に変更履歴を記録していく
  - ステージング = バージョン管理するファイルを指定(add)
  - コミット = ステージングしたファイルの変更を記録(commit)

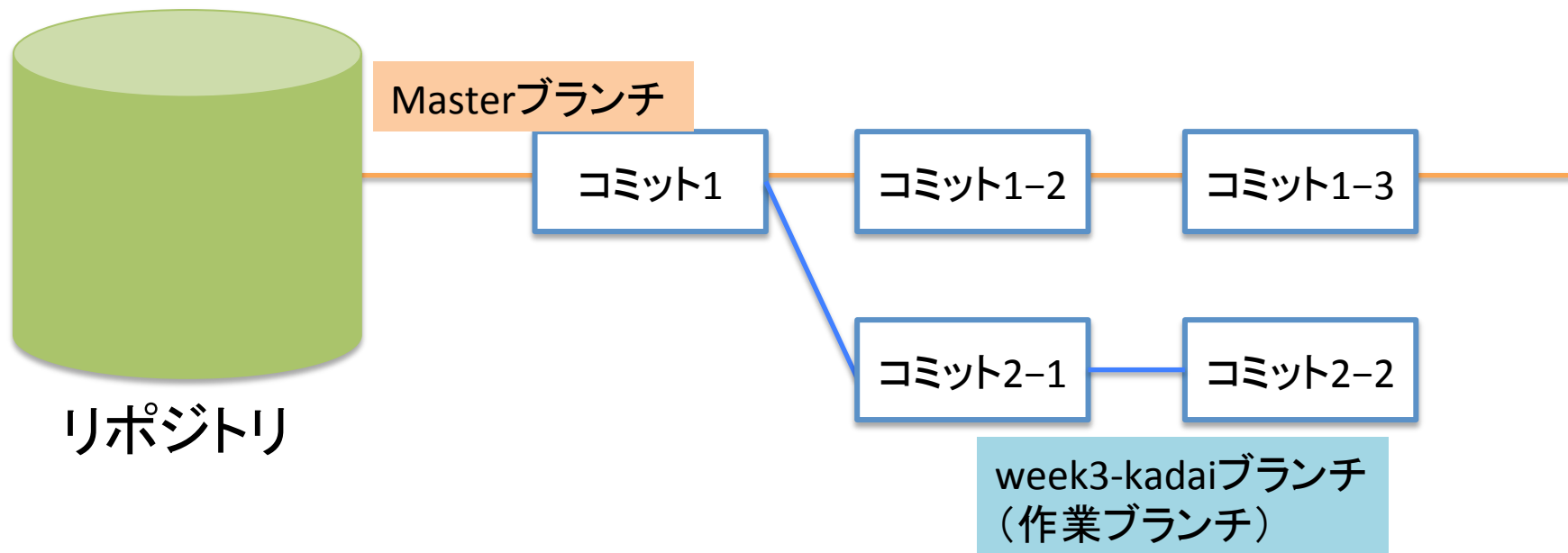


リポジトリ

- コミットすると変更内容、変更時間、コメント等が自動記録される

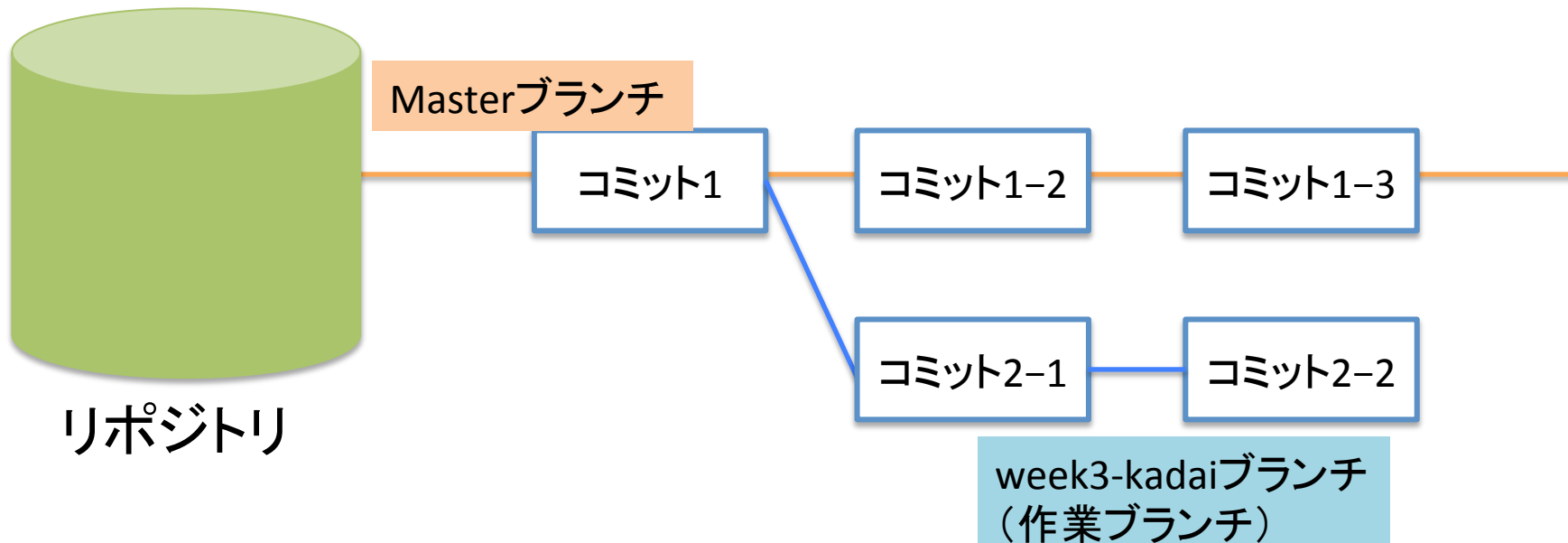
# ブランチとは？

- コミットを枝分かれさせる機能
  - ブランチ：分岐した枝のこと
  - Masterブランチ：デフォルトのブランチ
  - 作業ブランチ：Masterブランチ以外



# ブランチの利点

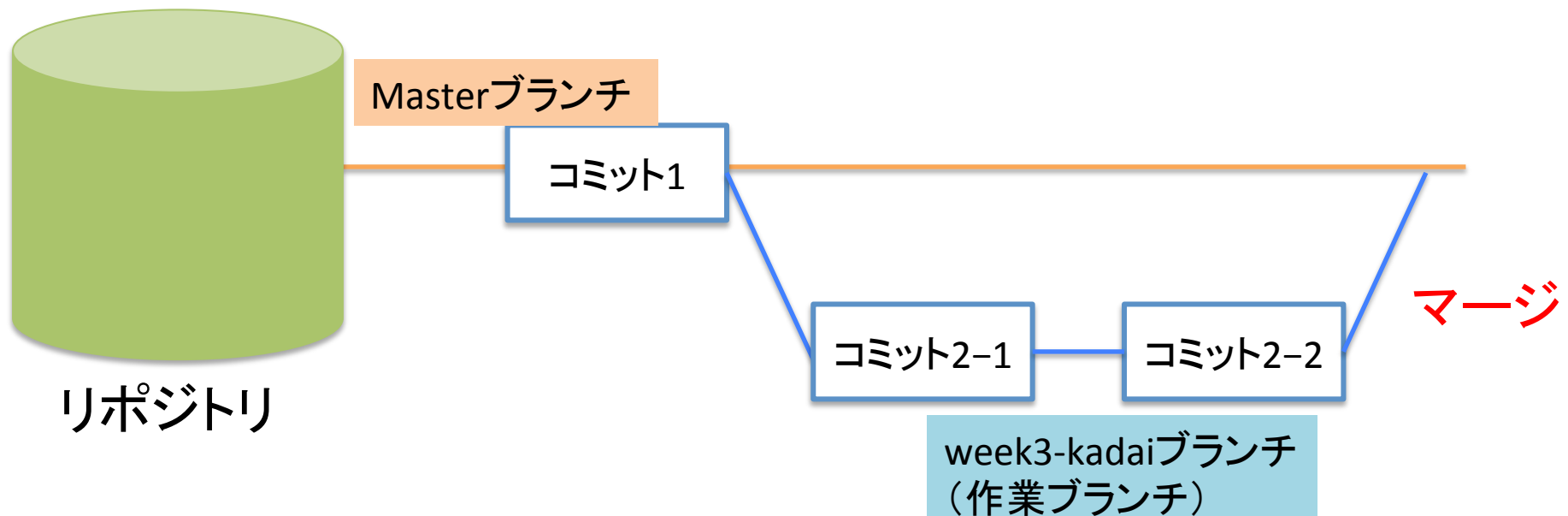
- ブランチを切り替えてコミットしても他のブランチは変更されない
  - 例) week3-kadaiブランチでコミットしてもMasterブランチの状態は変化しない
  - week3-kadaiブランチでエラーが取れなくなったらMasterブランチに戻れば元の状態を復元可能





# ブランチによる作業フロー

1. 新しい機能/新しい課題を行う際にブランチを作り、新しいブランチに移動
2. 新しいブランチで作業しadd/commit
3. 機能が完成したらMasterブランチにマージ



# Gitコマンドによるブランチ操作

- 新しいブランチの作成  
\$ git branch ブランチ名
- ブランチの切り替え  
\$ git checkout ブランチ名
- ブランチを一覧する  
\$ git branch
- ブランチをマージする  
\$ git checkout マージする先のブランチ名  
\$ git merge マージする元のブランチ名

# Gitコマンドによる作業フロー

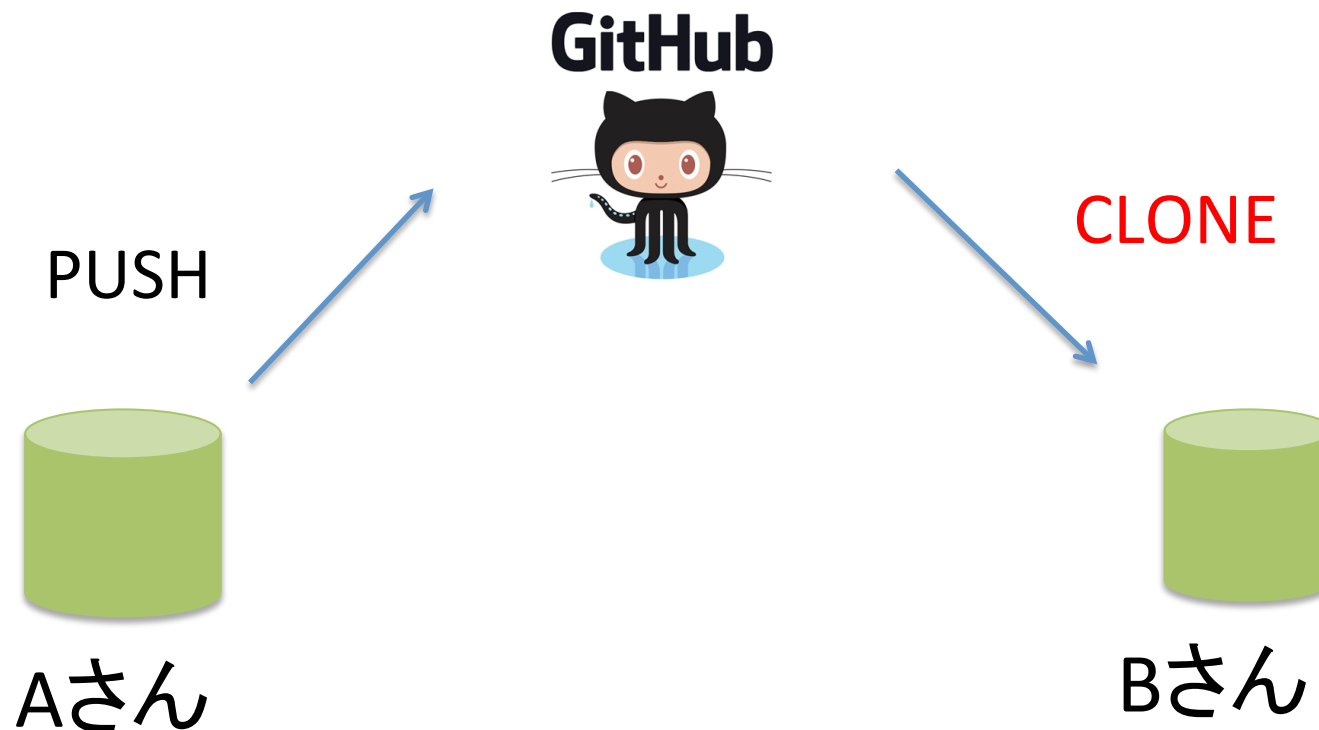
- 新しいブランチを切る
  - どのような作業をするのか分かるブランチ名を付けると良い
  - \$ git branch week3-kadai
- 新しいブランチに移動
  - ブランチの移動はgit checkoutコマンド
  - \$ git checkout week3-kadai
- 作業が終わったらマージする
  - まずはMasterに移動
  - \$ git checkout master
  - ブランチweek3-kadaiをMasterにマージ
  - \$ git merge week3-kadai

# 本日の講義

1. ブランチの基礎
2. バージョン管理システムを用いた  
ソースコードの配布
3. バージョン管理システムを用いた共同開発
4. Git以外のバージョン管理システム  
(分散型/集中型)

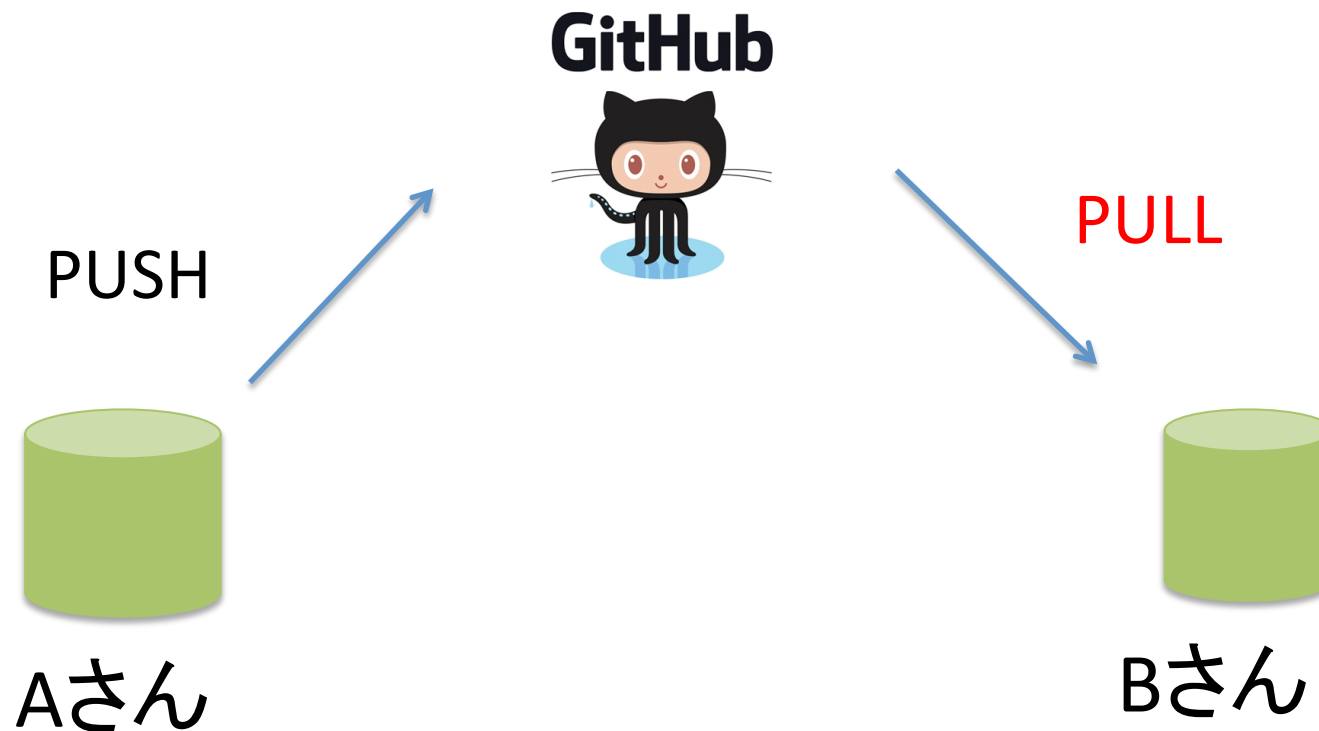
# 複数人で使う場合(ソースコード配布)

- AさんからBさんへコードを配布する
  - CloneするとBさんのローカルにAさんのリポジトリが複製される



# 複数人で使う場合(ソースコード更新)

- Aさんがさらにソースコードを更新しPUSH
- BさんはPULLすることでソースコードを最新の状態に出来る

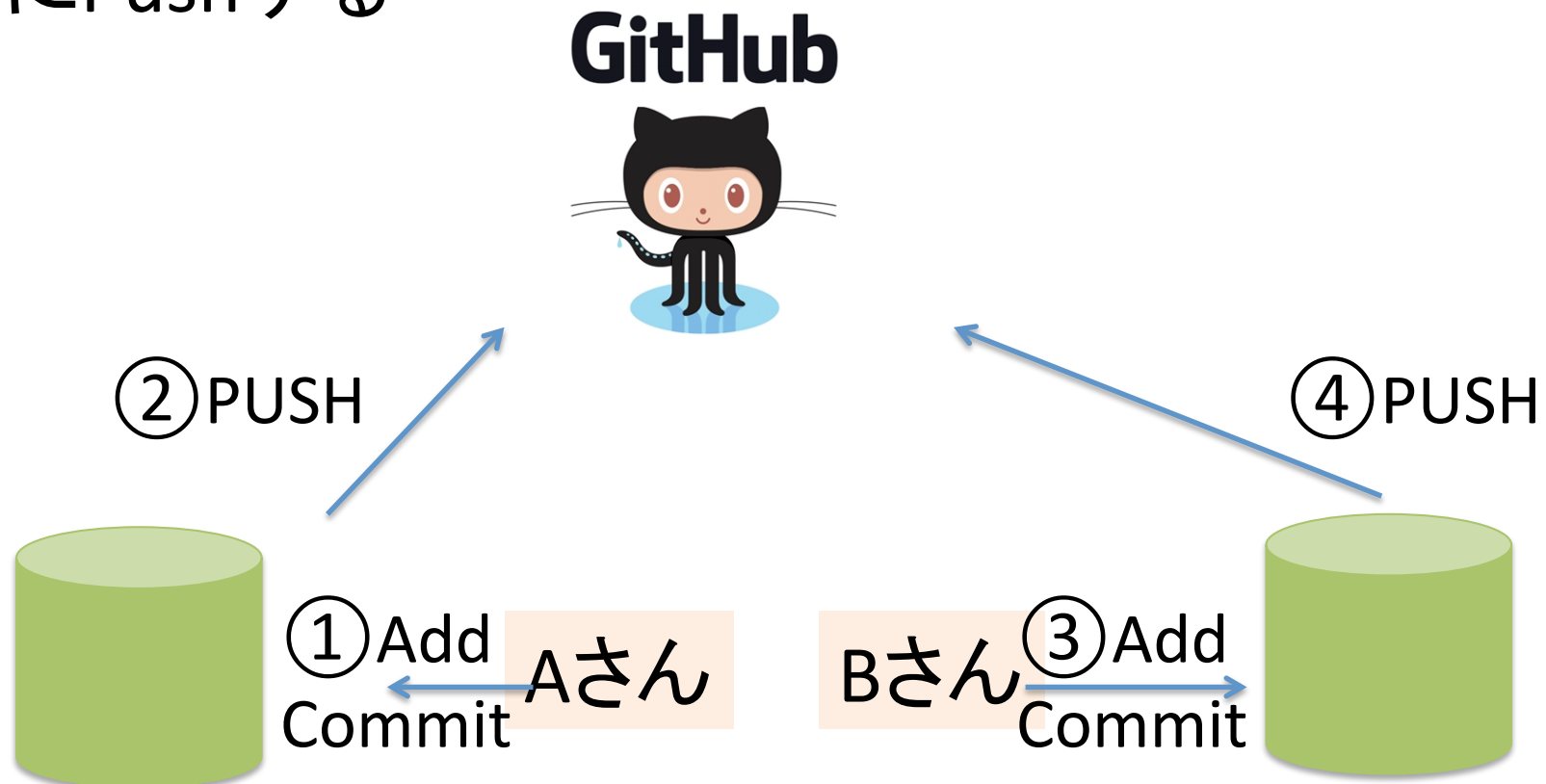


# 本日の講義

1. ブランチの基礎
2. バージョン管理システムを用いたソースコードの配布
3. バージョン管理システムを用いた共同開発
4. Git以外のバージョン管理システム  
(分散型/集中型)

# 複数人で使う場合

- 共同作業する場合はAさん、Bさんがそれぞれローカルリポジトリにadd/commitしてGitHubにPushする





# 補足:コンフリクト

- 複数人が同じファイルを編集すると競合(コンフリクト)が起こる
- 競合した場合、話し合ってコンフリクトを直すまでpushできない

# ソースコードと著作権(1/2)

- プログラムも著作物であれば著作権法の保護対象
  - 著作物 = 「思想又は感情を創作的に表現したものであって、文芸、学術、美術又は音楽の範囲に属するもの」
  - Hello Worldを表示するだけだと保護されないが、オリジナリティのあるプログラムは保護対象
  - ソフトウェアを構成する画像ファイルなども保護対象

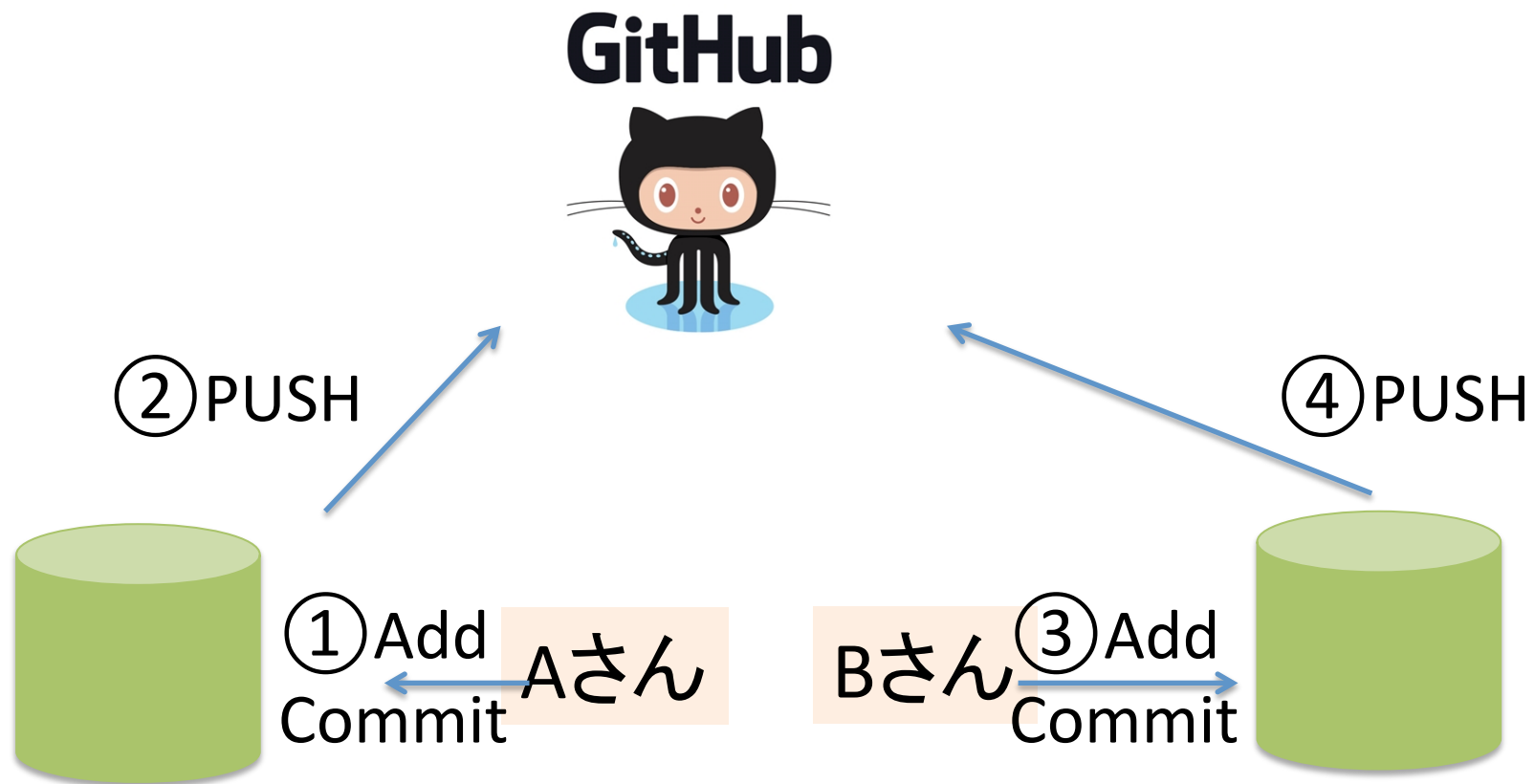
# 本日の講義

1. ブランチの基礎
2. バージョン管理システムを用いたソースコードの配布
3. バージョン管理システムを用いた共同開発
4. Git以外のバージョン管理システム  
(分散型/集中型)

# 分散型と集中型(1/2)

- Gitは**分散型**バージョン管理システム
  - 分散型の特徴：
    - ローカルリポジトリに一度コミットし履歴を残す
    - 好きなタイミングでリモートリポジトリにPUSHしソースコードを共有する
  - 分散型の利点：
    - インターネットに繋がっていない場合もコミットして履歴を残すことが出来る
  - 分散型の欠点：
    - ソースコードを共有するためにコミット/プッシュの2段階を踏む必要がありやや煩雑

# 分散型バージョン管理システム(図)



# 分散型と集中型(2/2)

- **集中型**バージョン管理システム

- 集中型の特徴:

- コミットすると直ちにリモートリポジトリに反映されソースコードが共有される
- Subversionが代表的なシステム

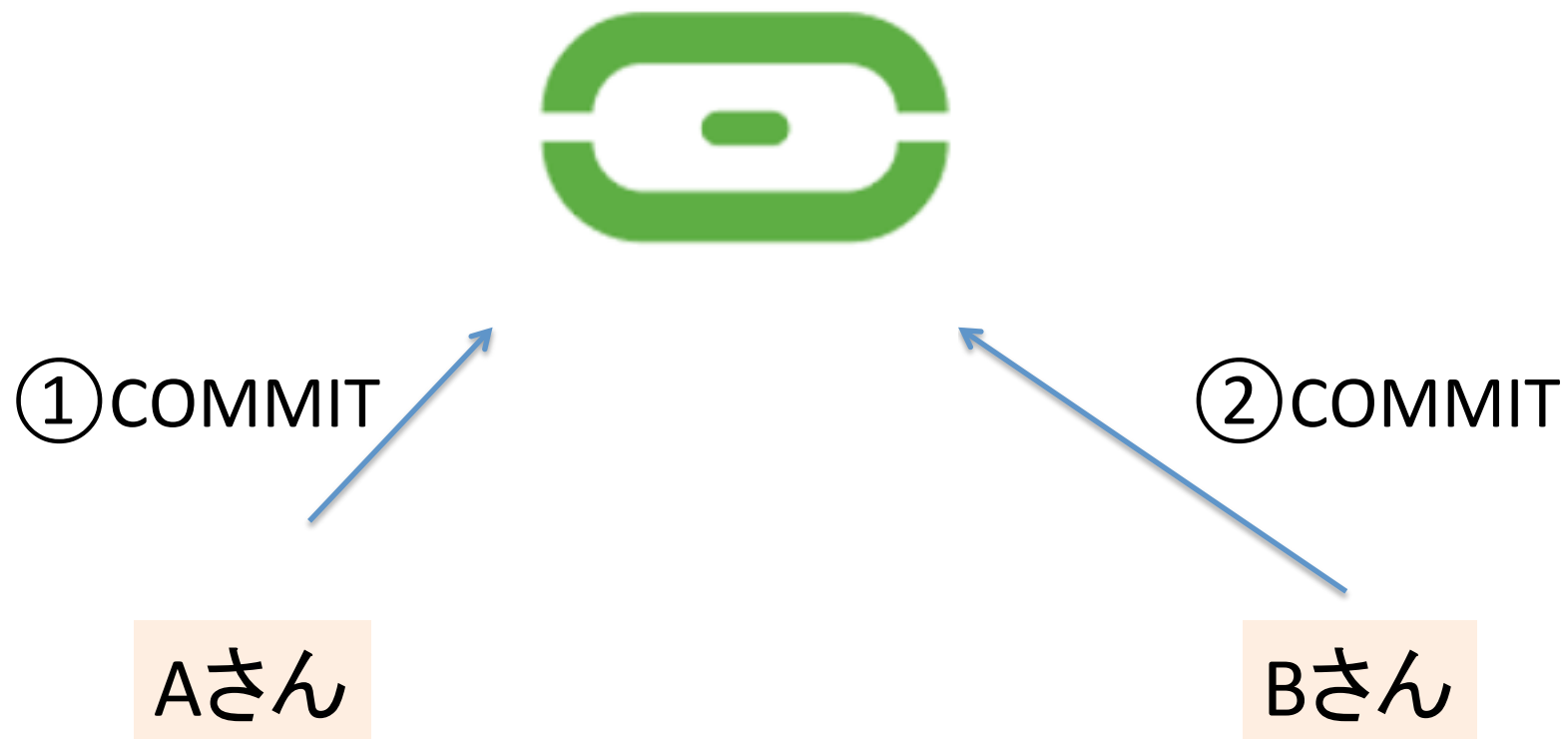
- 集中型の利点:

- コミットするだけでソースコードが共有でき操作は単純

- 集中型の欠点:

- コミットするためにインターネットへの接続が必須  
(大規模な共同開発には向かない)

# 集中型バージョン管理システム(図)



# 分散型と集中型の利点・欠点

- 分散型ではローカルリポジトリに一度Commitしてから、リモートリポジトリにPushする
  - 利点: インターネットに繋がっていない環境でもバージョン管理することが出来る
  - 欠点: Add/Commit/Pushという3段階でやや煩雑
- 集中型はリモートリポジトリにいきなりCommitする
  - 利点: add/commitするだけでリモートリポジトリに同期される
  - 欠点: オフラインでは使えない



# 演習0(役割分担)

- 隣の席の人と共同開発演習を行う
- 廊下側の人A、窓側の人Bと役割分担

# 演習0(課題用ブランチの作成)

1. 以下のgit branchコマンドで現在Masterブランチを使用していることを確認  
(使用中のブランチに\*マークが付く)  
\$ git branch
2. 以下のgit branchコマンドで演習用のブランチを作成  
\$ git branch week3-kadai
3. 再度git branchコマンドを入力し「week3-kadai」というブランチができていることを確認  
(week3-kadaiブランチが新たにできているが、\*マークはまだMasterに付いていることを確認)  
\$ git branch

# 演習1(ブランチの切り替え)

1. git checkoutコマンドで「week3-kadai」ブランチへ切り替え

```
$ git checkout week3-kadai
```

2. git branchコマンドで正しく移動できていることを確認

```
$ git branch
```

3. \*マークがweek3-kadaiに付いていることを確認し、次の演習に進む

# 演習2

- 集中型と分散型の違いについて100字程度でまとめる
- c-programming-practice-(学籍番号)以下にweek3というディレクトリを作成
  1. week3以下に「分散型と集中型.txt」というファイルを作成
  2. テキストエディタで違いについて100字程度で記述
    - 集中型の問題点や分散型を用いる利点について書くと良い


# 演習3(ブランチを指定してpush)

- 以下のコマンドで指定したブランチのみGitHubに提出

```
$ git add -A
```

```
$ git commit -m “分散型と集中型の違い(第3週)”
```

```
$ git push origin week3-kadai
```



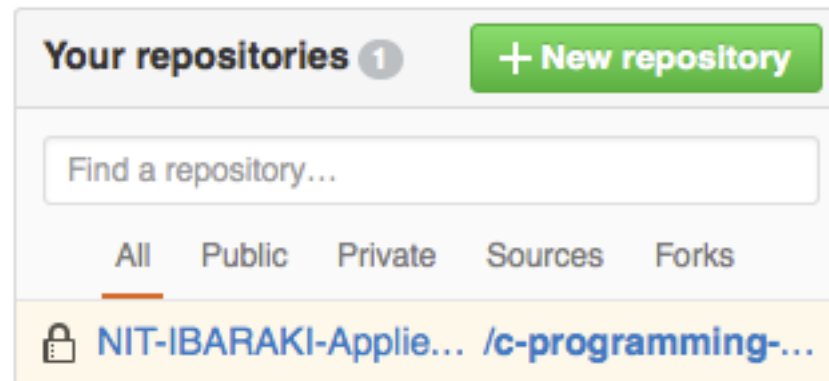
Originはリモートリポジトリの名前  
(GitHubが指定されている)



pushするブランチを指定

# 補足: GitHubにpushされたか確認

- <http://github.com>にアクセス
- 「Your repositories」の一覧から「c-programming...」を探してクリック



- commitsやbranchを押してみても正しくpushされているか確認



# 演習4(マージ)

1. git checkoutコマンドでMasterブランチに移動  
\$ git checkout master
2. git branchコマンドでMasterブランチに移動できたことを確認  
(\*マークがMasterについているか確認)  
\$ git branch
3. lsコマンドでファイル一覧を表示  
(Masterに移動するとweek3ディレクトリが消えるか確認)  
\$ ls
4. git mergeコマンドでweek3-kadaiブランチをMasterブランチにマージ  
(かならずMasterブランチ上で操作すること！)  
\$ git merge week3-kadai
5. 再度lsコマンドでファイル一覧を表示し、week3ディレクトリが無事にマージされたか確認

- 早く終わった人
  - 先々週のC言語課題のやり残しがあればやっておく
  - ブランチを他にも作ってMasterブランチにMergeしてみよう
  - 周りの人に教えてあげる
  - 次週はUNIXコマンドの基礎を学びます。