

First step for version control

# Introduction for Git

s1220233

Ryoya Komatsu

# 自己紹介

- ◎ 名前 : 小松 凌也
- ◎ Slack ID : @michael
- ◎ 学年 : 学部 3 年
- ◎ 出身地 : 福島県棚倉町
- ◎ 好きな言語 : Python(2.x)

# 目次

- ◎ 導入
- ◎ Gitって何？
- ◎ Gitの構造と仕組み
- ◎ Gitの使い方
- ◎ 演習問題
- ◎ 参考資料



# 導入

みなさんはGitを  
どの程度理解していますか？

# 導入

- ◎ 「add→commit→push」 ってやると  
GitHubにファイルが打ち上げられる
  - ◎ 言われるがまま思考停止して使っている
  - ◎ 難しいし意味分らない
  - ◎ コマンドとかオプション多すぎ
  - ◎ めんどい
- 
- ◎ GitHubにファイル打ち上げてどうすんの？

同感です。

# 導入

- ◎ Gitとは何なのかを学ぶ前に、そもそも「バージョン管理システムとは何なのか」を知る必要がある。



# 導入 - バージョン管理システムとは

## バージョン管理システム (Version Control System, VCS)

- ◎ ファイルの変更履歴をバージョン情報として蓄積して、ファイルに関する5W1Hの情報群を容易に管理できるようにするための便利ツール

※5W1H ... when, where, who, what, why, how

..... ? ? ?

# 導入 - バージョン管理システムとは

The image shows two side-by-side terminal windows. The left window displays the following C code:

```
1 #include <stdio.h>$
2 $
3 int main(){$
4     int a = 100;$
5 $
6     printf("10 squared is %d !\n", a);$
7 $
8     return 0;$
9 }$
```

The status bar at the bottom of the left window shows: <MAL R0 | test.c unix | utf-8 | c 11% 1:1 "test.c" [readonly] 9L, 106C

The right window displays the following C code:

```
1 #include <stdio.h>$
2 $
3 int main(){$
4     int a, b;$
5 $
6     scanf("%d", &a);$
7     b = a * a;$
8 $
9     printf("%d squared is %d !\n", a, b);$
10 $
11     return 0;$
12 }$
```

The status bar at the bottom of the right window shows: NORMAL test.c unix | utf-8 | c 8% 1:1 "test.c" 12L, 143C

## 変更前の test.c と変更後の test.c

- 4行目にint bの変数宣言が追加されている
- scanfで数字を読み取り、その二乗を求める処理が追加されている
- それらを表示するprintfが少し書き換えられている

# 導入 - バージョン管理システムとは

- ◎ 「数字の二乗を表示する」というプログラム自体の目的は変わっていないが、変更の前後でその機能が変わっている。

→ 変更前をver1.0とすると変更後はver2.0

# 導入 - バージョン管理システムとは

- ◎ VCSを使うと、「いつ・誰が・どこに・どんな変更を加えたのか」を全て保存することができる。
- ◎ 変更して上書き保存してしまった場合でも、過去のバージョン情報をVCSを使って記録しておけばver2.0のtest.cをver1.0に(更にその前のバージョンにも)戻すことができる。

# 導入 - バージョン管理システムとは

更に...

- 複数人でチームを組んで一つのソフトウェアを開発するときにとっても便利。
- GitHubなどと連携すればソースコードのバックアップも同時にこなせるのでとても便利。

結論：めっちゃ便利

Gitって何？



# Gitって何？

- ◎ 分散型バージョン管理システム  
(Distributed Version Control System, DVCS)  
の一つ
- ◎ ディレクトリの中のファイルの変更履歴  
を保存し、VCS用の共有ウェブサービ  
スなどと連携しソフトウェアのバージョ  
ンを管理するためのツール

# Gitって何？

- **git**（ギット）は、プログラムのソースコードなどの変更履歴を記録・追跡するための分散型バージョン管理システムである。Linuxカーネルのソースコード管理に用いるためにリーナス・トーバルズによって開発され、それ以降ほかの多くのプロジェクトで採用されている。

- Wikipediaより引用

# Gitって何？

- ◎ GitHubやBitBucketなどのVCS用共有ウェブサービスなどと連携することによって、自分の手元にあるプロジェクトの開発ディレクトリを瞬時にチームメンバーと共有することができ、進捗状況の同期や復元を容易に行うことができる。



# Gitって何？

## 大切なこと

- ◎ Gitはバージョン管理用のツール、GitHubはそれを利用したサービスの一つです。  
別にGitとGitHubが2つで一つだというわけではなく、それぞれ独立しているものだというのをしっかり覚えておきましょう。

# Gitの構造と仕組み

# Gitの構造と仕組み

Gitを学ぶ上で重要なキーワード

- ◎ ワーキングディレクトリ
- ◎ リポジトリ(ローカル、リモート)
- ◎ ステージングエリア

# Gitの構造と仕組み

## ◎ ワーキングディレクトリ

ユーザーがファイル内容の編集や削除などの作業をする場所。ワーキングディレクトリ内のファイルの状態は常にGitによって監視されているが、ユーザーがワーキングディレクトリの存在を特に意識する必要はない。

# Gitの構造と仕組み

- ◎ ローカルリポジトリ、リモートリポジトリ

変更履歴の情報を保存しておくための場所をリポジトリという。Gitは分散型のバージョン管理システムなので、ワーキングディレクトリ内(ローカル)と共有サービス上(リモート)にリポジトリを持つ。



# Gitの構造と仕組み

- ◎ ローカルリポジトリ、リモートリポジトリ

ローカルリポジトリはワーキングディレクトリ内に隠されているが、変更記録の保存などは全てGitが見えない所で勝手に行ってくれるので特に気にする必要はない。

# Gitの構造と仕組み

## ◎ ステージングエリア

変更をリポジトリに記録するときに、どのファイルの変更を記録するか区別するためのスペース。

# Gitの構造と仕組み

## ◎ ステージングエリア

ワーキングディレクトリのファイルを変更しただけではリポジトリに変更が保存されない。



変更したファイルをステージングエリアに追加し、これから変更を記録することを予めGitに知らせておく。

# Gitの構造と仕組み

## Gitの基本フロー

- ◎ 「ワーキングディレクトリ内」のファイルを編集する
- ↓
- ◎ 編集したファイルを「ステージングエリア」に追加する
- ↓
- ◎ 「ローカルリポジトリ」にコミットする
- ↓
- ◎ 「リモートリポジトリ」にプッシュする

# Gitの構造と仕組み

ローカル(自分のPC)環境上の構成

1. ワーキングディレクトリ
  2. ローカルリポジトリ
  3. (ステー징エリア)
- ◎ ステージングエリアは概念のようなものなので、実際にユーザーから可視的であるわけではない。

# Gitの構造と仕組み

## ローカル(自分のPC)環境上の構成

- リポジトリに対してファイルの変更を保存する操作自体は、ローカル環境のみで完結している。

→ ローカルリポジトリへのコミットの度にリモートリポジトリにプッシュする必要はなく、複数のコミットを重ねた任意のタイミングでプッシュすることができる。

ちょっと難しい話

# Gitの構造と仕組み

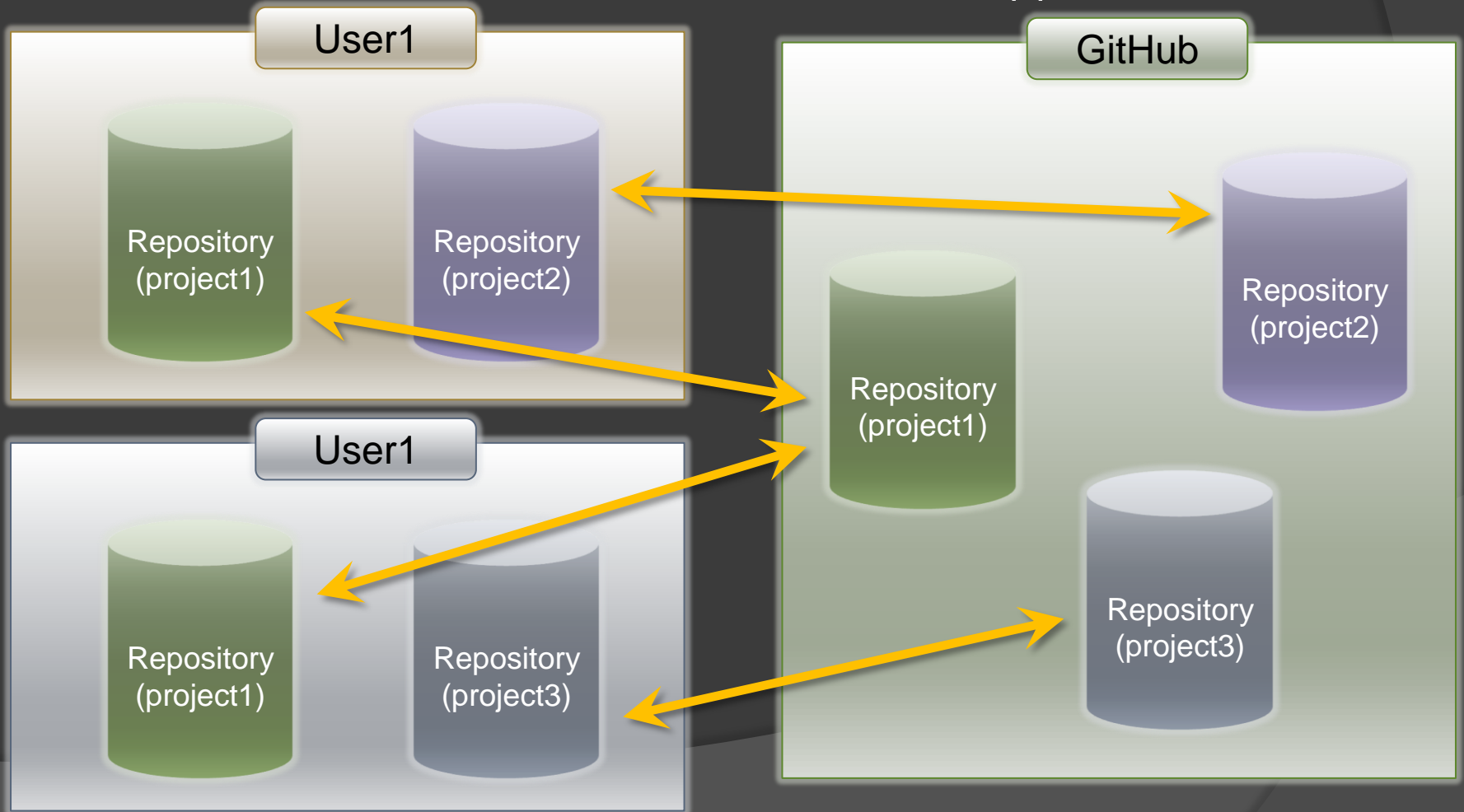
## ローカルとリモートの関係

- ◎ （GitHubなどを利用している場合） リモートリポジトリは一つのプロジェクトにつきGitHub上に一つ置かれる。
- ◎ ローカルリポジトリは、プロジェクトに参加している各ユーザーのマシン上に置かれる。



# Gitの構造と仕組み

## ローカルとリモートの関係



# Gitの構造と仕組み

## ローカルとリモートの関係

- オリジナルはリモートリポジトリ（中央リポジトリ）であり、各ユーザーの持つローカルリポジトリはそのリモートリポジトリのコピーであるという考え方

# Gitの構造と仕組み

## Gitのライフサイクル

- Gitで管理されているディレクトリは、

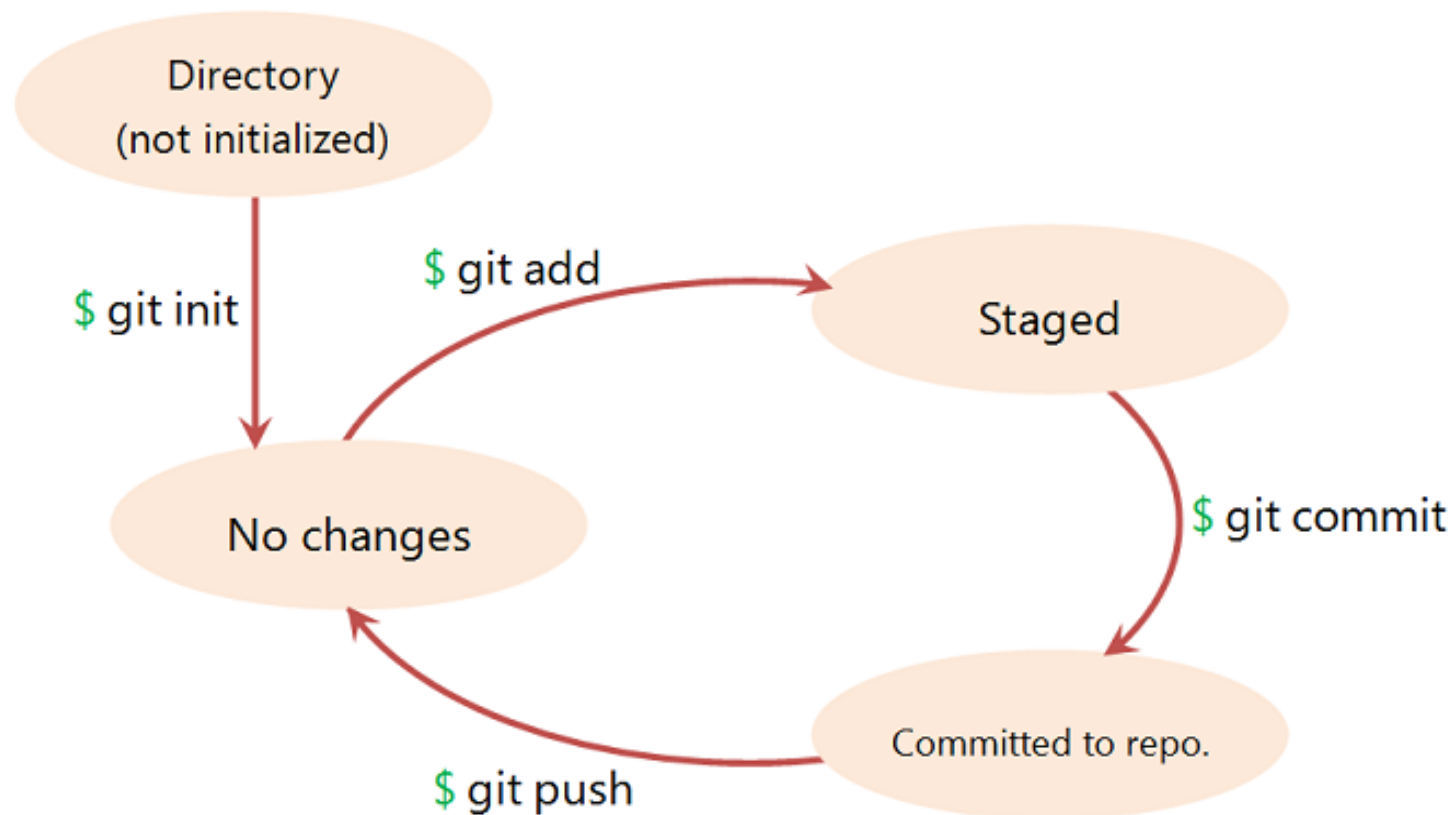
**add – commit – push**

を一つのサイクルとして状態が循環的に変化していく。

慣れないうちは、上のサイクルを一連のコマンドの塊として覚えてしまっても良い。

# Gitの構造と仕組み

## Gitのライフサイクル



# Gitの構造と仕組み

## Gitのライフサイクル

- ◎ 課題や演習問題の提出は、「リモートリポジトリへのプッシュ」を提出したものとみなします。
  - コードは完成したけどaddしてない
  - addしたけどcommitしてない
  - ローカルにcommitしたけどリモートにpushしてない
- などといったことがないように、  
きちんと **add – commit – push** しましょう。

# Gitの使い方