# Git 講座

# はじめに

- Git は奥が深い
- 私も4割しか理解していない
- 今日は基礎的なことしかやりません

- やること: コミット・プッシュ
  チェックアウト (ブランチの概念), 歴史など

# 用語

- ディレクトリ: フォルダのこと.

# Git とは

- 2005年，リーナス・トーバルズというプログラマが作った **分散型バージョン管理システム**
  - リーナス・トーバルズ: Linux という OS を作ったおじさん
  - 今は濱野純 (はまのじゅん) という Google のエンジニアが管理しています

# Git の核となる概念

# リポジトリ

- プロジェクトの全履歴を保存する「倉庫」

- git init → 新規のリポジトリを作成
- git clone → 既存のリポジトリを複製

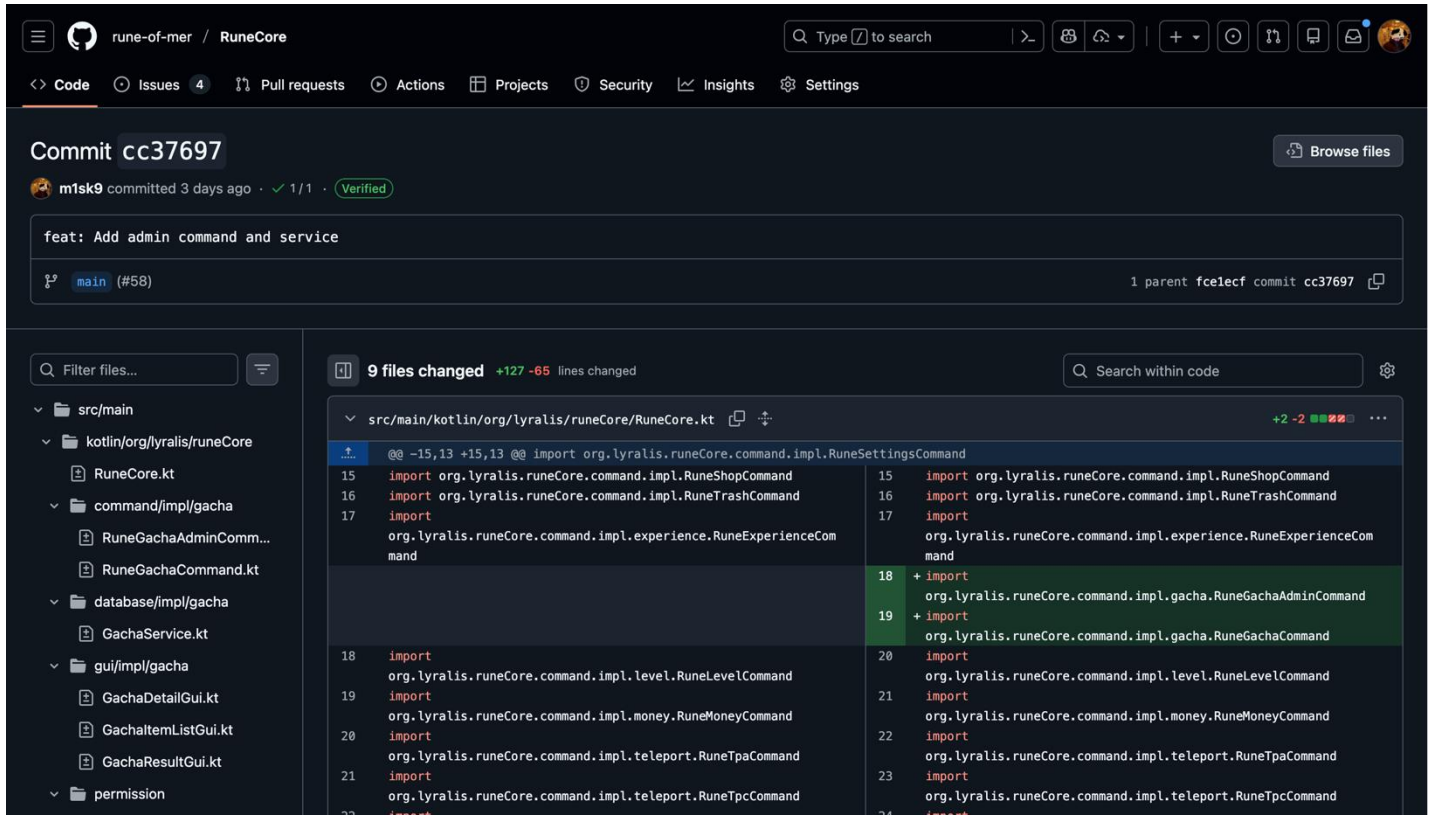
# ４つの領域

- 作業ディレクトリ→ステージング
  →ローカルリポジトリ →リモートリポジトリ

- 作業ディレクトリ: 実際にファイルを編集する場所
- ステージング: 次のコミットに含める準備を行う場所
- ローカルリポジトリ: 自分のPC内の履歴のデータベース
- リモートリポジトリ: GitHub などのサーバ上にある
  共有データベース

# コミット

- ある時点の「スナップショット」を示すもの
- 各コミットには
  - 変更内容
  - 作者と日時データ
  - コミットメッセージ
  - 一意のハッシュ値



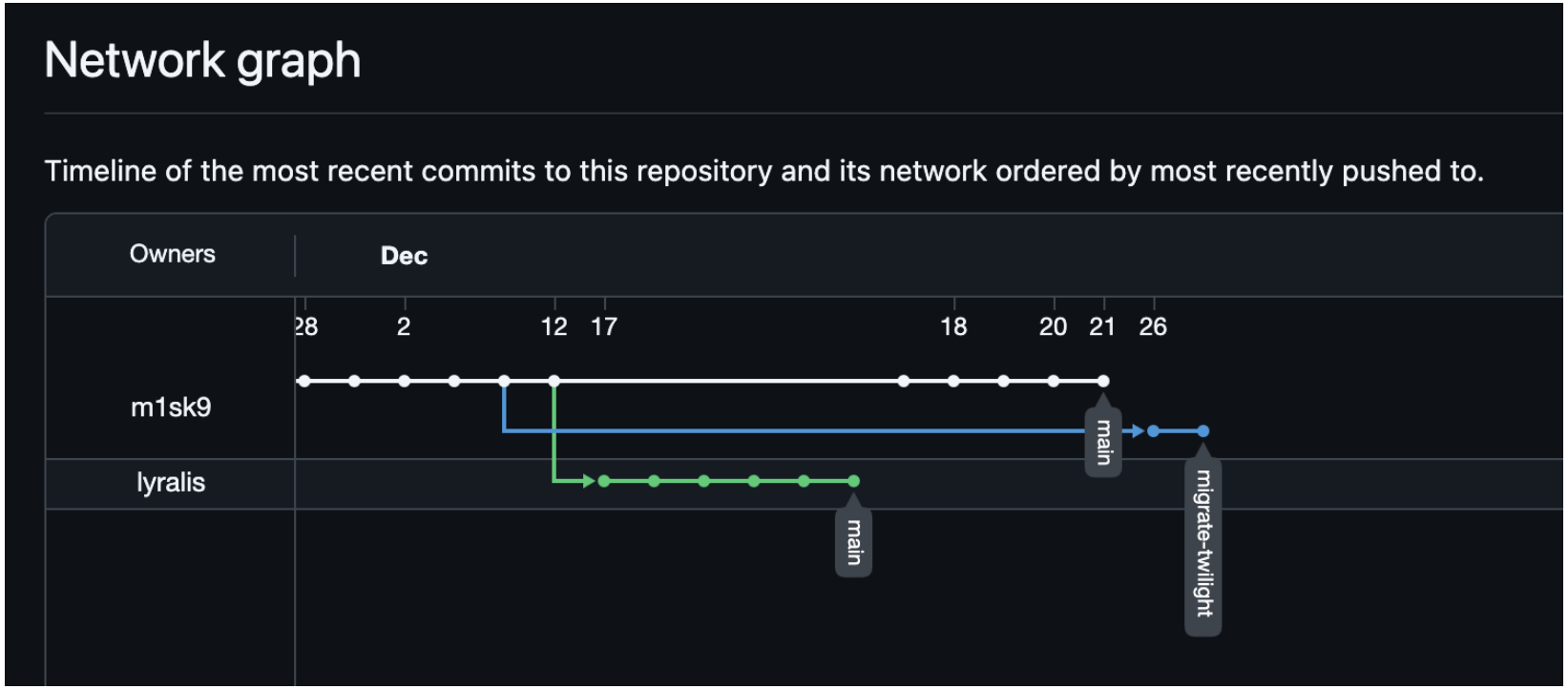
# コミット/プッシュ

- GitHub に変更をアップロードするには コミット・プッシュが必要

- git add <ファイル名>, git add . → 変更をステージングへ
- git commit –m <メッセージ> → コミットする
- git push origin main → リモートリポジトリにプッシュする

# ブランチの考え方

- Git には「コミット履歴の分岐点」である「ブランチ」という考え方がある



# ブランチ

- “main” ブランチ→メインブランチであり，「動く状態」であることが求められる

- Git はこのブランチを無限に生やすことができる
- この “main” を維持したまま 「新機能の実装」や「バグ修正」などができ，うまくいかなければいつでも削除ができる

# 並行作業もできる

- 無限に作成できるので複数の作業を同時に進行できる

```
m1sk9@dev-m1sk9-lap1 ~/R/g/r/RuneCore (feat/Add-ability-weapon)> git branch
  chore/add-claude-file
  chore/update-docs-and-debug-server
  claude/add-notification-queue-TmaMf
  claude/implement-gacha-system-2NgLn
  claude/player-settings-system-3Gc2B
  docs/add-database
  docs/deploy-javadoc
* feat/Add-ability-weapon
  feat/add-command-system
  feat/add-more-money-system
  feat/custom-item
  feat/debug-mode
  feat/gui
  feat/level-system
  feat/menu
  feat/money
  feat/setup-db
  feat/shop
  feat/teleport
  feat/world
  fix/database-uuid-type
  fix/deploy-javadoc-workflow
  main
  perf/debug-server
  rename-package
  renovate/mariadb-12.x
m1sk9@dev-m1sk9-lap1 ~/R/g/r/RuneCore (feat/Add-ability-weapon)>
```

# マージ

- ブランチを 「統合」する
- リモートリポジトリの変更を統合する場合はプルリクエストというものを作成する
  - プルリクエストは GitHub での名称であって，それ以外のサービス (Bitbucket や GitLab) などではまた別の名称で呼ばれてる

# コンフリクト

- 全ての状態で完璧に「統合」するのは難しく，変更箇所が重なっていると「コンフリクト」という状態に陥る
- このコンフリクトの解消が Git を使う上で特に難しい難関とも言える場所でもある

# 基本はコマンドライン

- Git はコマンドラインからの操作が基本となる
- でもコマンドが複数あって覚えるのが難しい

```
m1sk9@dev-m1sk9-lap1 ~> git --help
usage: git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--no-lazy-fetch]
           [--no-optional-locks] [--no-advice] [--bare] [--git-dir=<path>]
           [--work-tree=<path>] [--namespace=<name>] [--config-env=<name>=<envvar>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   restore    Restore working tree files
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   diff       Show changes between commits, commit and working tree, etc
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   backfill   Download missing objects in a partial clone
   branch     List, create, or delete branches
   commit     Record changes to the repository
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   reset      Reset current HEAD to the specified state
   switch     Switch branches
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
See 'git help git' for an overview of the system.
m1sk9@dev-m1sk9-lap1 ~>
```

# GUI 環境(画面上) からでの Git 操作

- JetBrains IDE (IDEA) や VSCode は Git 対応が進んでる
- 画面上からのクリックだけで簡単にコミットやプッシュができる
- ただ，この操作に慣れてしまってコマンドラインでの操作ができないのは本末転倒で，エンジニアを目指しているのなら尚更コマンドラインに慣れておかないといけない

# Progate というサービス

- Progate というプログラミング学習サービスに Git コースがあります
- 私は使ったことがないのでわかりやすいかどうかはわかりませんが，このパワーポイントよりは流石にわかりやすいと思います

https://prog-8.com/courses/git

# とりあえず

- コミット，プッシュ，チェックアウト，マージ，プルくらいは覚えておきましょう

# 使うな

- Git を使う上で使わないほうがいいコマンドを教えます

別にコマンド自体は使うし，便利です
何が起きるか十二分に理解していない状態で使わないほうがいいです

- git push -f , git push --force
- git reset --hard
- git rebase (これを僕たちは歴史改変と呼んでいます)

# Git を使う上で守っておいてほしいこと

1. コミットメッセージはわかりやすいものにしましょう
   - ”直した” とか “新しい機能” とかふざけたメッセージは避けます
   - 使ったらしばく
2. わからなければ聞くこと
   - Git は難しく，操作をミスると履歴が破壊されたり，修復困難になりがちです (私も何回も履歴を破壊して他人を困らせました)
   - わからなければ同じチームの人などに聞きましょう