Git Bash 完全マスターガイド

🚀 第1章: Git Bashの立ち上げ方

方法1: デスクトップから起動

1. デスクトップの何もないところで「右クリック」
2. 「Git Bash Here」をクリック
   ↓
Git Bashが開く!

方法2: フォルダから起動(おすすめ!)

1. 作業したいフォルダを開く
   例: C:\projects\my-project
2. フォルダ内で「右クリック」
3. 「Git Bash Here」をクリック
   ↓
そのフォルダで Git Bash が開く!

方法3: スタートメニューから

1. Windowsキー押す
2. 「Git Bash」と入力
3. 「Git Bash」をクリック
   ↓
ホームディレクトリで開く

方法4: VSCodeから起動

1. VSCodeでプロジェクトを開く
2. メニューバー「Terminal」→「New Terminal」
3. 右上の「+」横のドロップダウン
4. 「Git Bash」を選択

起動後の画面

User@PC-NAME MINGW64 ~
$

意味:

- `User` = あなたのユーザー名
- `PC-NAME` = コンピューター名
- `MINGW64` = Git Bashの環境
- `~` = ホームディレクトリ(現在地)
- `$` = コマンド入力待ち

📂 第2章: プロジェクトフォルダへの移動

基本コマンド

```
# 現在地を確認
pwd
# 読み方: ピー・ダブリュー・ディー
# 意味: print working directory = 今いる場所

# ファイル一覧を見る
ls
# 読み方: エル・エス
# 意味: list = 一覧表示

# 詳細表示
ls -la
# 読み方: エル・エス -エル・エー
```

フォルダ移動

```
# Cドライブのprojectsフォルダへ移動
cd /c/projects
# 読み方: シー・ディー /シー/プロジェクツ

# プロジェクトフォルダへ
cd /c/projects/my-project

# 1つ上の階層へ
cd ..

# ホームディレクトリへ
cd ~
```

パスの書き方:

```
Windows: C:\projects\my-project
Git Bash: /c/projects/my-project
        ↑ スラッシュに注意!
```

🔍 第3章: ローカルとリモートの状態確認

1. ローカルの状態を見る

```
# 現在のブランチと変更状況
git status
```

表示例:

```
On branch feature/my-work
Your branch is ahead of 'origin/feature/my-work' by 2 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  modified:   login.py

Untracked files:
  new-file.py
```

読み方:

- `On branch feature/my-work` = 今このブランチにいる
- `ahead by 2 commits` = ローカルが2コミット進んでいる(プッシュしていない)
- `modified: login.py` = 変更されたファイル
- `Untracked files` = まだGit管理していないファイル

---

2. ブランチを見る

```
# ローカルのブランチ一覧
git branch

# 表示例:
  main
* feature/my-work    ← * は今いるブランチ
  feature/old-work

# リモートのブランチも含めて全部表示
git branch -a

# 表示例:
  main
```

```
* feature/my-work
  remotes/origin/main
  remotes/origin/feature/my-work
  remotes/origin/feature/other-person
```

色の意味:

- 緑 = 今いるブランチ
- 白 = ローカルの他のブランチ
- 赤 = リモートのブランチ

---

3. リモートリポジトリの確認

```
# リモートリポジトリのURL確認
git remote -v
```

表示例:

```
origin  https://github.com/username/my-project.git (fetch)
origin  https://github.com/username/my-project.git (push)
```

意味:

- `origin` = リモートの別名(デフォルト名)
- `fetch` = ダウンロード用URL
- `push` = アップロード用URL

```
# 詳しい情報
git remote show origin
```

表示例:

```
* remote origin
  Fetch URL: https://github.com/username/my-project.git
  Push  URL: https://github.com/username/my-project.git
  HEAD branch: main
  Remote branches:
    main              tracked
    feature/my-work   tracked
  Local branch configured for 'git pull':
    feature/my-work merges with remote feature/my-work
  Local ref configured for 'git push':
    feature/my-work pushes to feature/my-work (up to date)
```

---

4. ローカルとリモートの差を確認

まずリモートの最新情報を取得
git fetch origin

ローカルとリモートの差分(コミット数)
git status

表示例:
Your branch is ahead of 'origin/main' by 3 commits.
↑ ローカルが3コミット進んでいる(プッシュしてない)

Your branch is behind 'origin/main' by 2 commits.
↑ リモートが2コミット進んでいる(プルしてない)

Your branch is up to date with 'origin/main'.
↑ 同期している

---

5. コミット履歴で確認

ローカルのコミット履歴
git log --oneline -5

表示例:
abc1234 (HEAD -> feature/my-work) ログイン機能追加
def5678 バグ修正
ghi9012 初期設定
jkl3456 (origin/main, main) READMEを更新
mno7890 プロジェクト初期化

記号の意味:

- `HEAD` = 今いる場所
- `origin/main` = リモートのmainブランチ
- `main` = ローカルのmainブランチ

グラフで見やすく
git log --oneline --graph --all -10

表示例:

* abc1234 (HEAD -> feature/my-work) ログイン機能追加

```
* def5678 バグ修正
| * xyz1111 (origin/feature/other) 他の人の作業
|/
* ghi9012 (origin/main, main) 初期設定
```

🔥 第4章: コンフリクトの差分の見方

コンフリクト発生時

```
$ git pull origin main

Auto-merging login.py
CONFLICT (content): Merge conflict in login.py
Automatic merge failed; fix conflicts and then commit the result.
```

Step 1: コンフリクトしているファイルを確認

```
git status
```

表示:

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   login.py
        both modified:   config.py
```

Step 2: 差分を見る

```
# コンフリクトファイルの差分
git diff login.py
```

表示例:

```
++<<<<<<< HEAD
 +    if user.authenticate(password):
 +        return True
++||||||| merged common ancestors
+     if user.check_password(password):
+         return True
++=======
+     if user.verify_password(password):
+         session.create(user)
+         return True
```

++>>>>>>>> origin/main

意味:

<<<<<<< HEAD
↑ ここから下が「あなたのローカル」の変更

||||||| merged common ancestors
↑ 元のコード(共通の祖先)

=======
↑ 区切り線

>>>>>>> origin/main
↑ ここまでが「リモート」の変更

Step 3: 3方向差分で見る(詳しく)

```
git diff --ours login.py      # 自分の変更だけ
git diff --theirs login.py    # 相手の変更だけ
git diff --base login.py      # 元のコードとの差分
```

Step 4: VSCodeで見る(超おすすめ!)

```
# VSCodeで開く
code login.py
```

VSCodeの表示:

```
<<<<<<< HEAD (Current Change)
    if user.authenticate(password):
        return True
=======  (Incoming Change)
    if user.verify_password(password):
        session.create(user)
        return True
>>>>>>> origin/main
```

[Accept Current Change] [Accept Incoming Change]
[Accept Both Changes] [Compare Changes]

ボタンをクリックするだけで解決!

Step 5: コンフリクト解決後の確認

```
# 解決したか確認
git status

# まだコンフリクト中:
Unmerged paths:
  both modified:   login.py

# 解決済み(addした後):
Changes to be committed:
  modified:   login.py
```

📊 第5章: 2つのコミット間の差分

コミットIDの確認

```
# コミット履歴を表示
git log --oneline

# 表示例:
abc1234 最新のコミット
def5678 1つ前
ghi9012 2つ前
jkl3456 3つ前
mno7890 4つ前
```

パターン1: 2つのコミットIDを指定

```
# コミットID間の差分
git diff ghi9012 abc1234

# 読み方: 古いID 新しいID
```

短縮形でもOK:

```
git diff ghi9012 abc1234
git diff ghi901 abc123    # 最初の6-7文字でOK
```

パターン2: 相対的な指定

```
# 最新と1つ前の差分
git diff HEAD~1 HEAD

# 最新と3つ前
git diff HEAD~3 HEAD
```

```
# HEAD~1 = 1つ前
# HEAD~2 = 2つ前
# HEAD~3 = 3つ前
```

パターン3: 変更されたファイルだけ見る

```
# ファイル名のみ
git diff --name-only ghi9012 abc1234

# 表示:
login.py
config.py
app.py

# 統計情報
git diff --stat ghi9012 abc1234

# 表示:
 login.py  |  5 +++--
 config.py | 12 +++++++++---
 app.py    |  3 +--
 3 files changed, 15 insertions(+), 5 deletions(-)
```

パターン4: 特定ファイルの差分だけ

```
git diff ghi9012 abc1234 -- login.py

# 読み方: git diff 古いID 新しいID -- ファイル名
```

パターン5: コミットとHEADの差分

```
# 特定のコミットと現在の差分
git diff ghi9012

# これは以下と同じ意味
git diff ghi9012 HEAD
```

パターン6: ブランチ間の特定コミット範囲

```
# mainブランチの2つのコミット間
git diff main~3..main~1

# feature/my-workブランチの最新3コミット
git diff HEAD~3..HEAD
```

👤 第6章: 特定の人の変更だけ見る方法

方法1: git logで人を指定

```
# 特定の人のコミットだけ表示
git log --author="Tanaka"

# 読み方: ギット ログ --オーサー="タナカ"
```

表示例:

```
commit abc1234
Author: Tanaka Taro <tanaka@example.com>
Date:   Mon Oct 6 14:30:00 2025

    ログイン機能を追加

commit def5678
Author: Tanaka Taro <tanaka@example.com>
Date:   Mon Oct 6 10:00:00 2025

    バグ修正
```

```
# メールアドレスでも検索可能
git log --author="tanaka@example.com"

# 部分一致でOK
git log --author="tana"
```

方法2: 特定の人の変更ファイル一覧

```
# Tanakaさんが変更したファイル
git log --author="Tanaka" --name-only --oneline
```

表示:

```
abc1234 ログイン機能を追加
login.py
auth.py

def5678 バグ修正
app.py
```

方法3: 特定の人の差分を見る

```
# Tanakaさんの最新コミットの差分
git log --author="Tanaka" -1 -p

# -1 = 最新1件
# -p = 差分も表示
```

表示:

```
commit abc1234
Author: Tanaka Taro <tanaka@example.com>

    ログイン機能を追加

diff --git a/login.py b/login.py
+++ b/login.py
@@ -10,3 +10,8 @@
+def authenticate(username, password):
+    user = User.find(username)
+    return user.check_password(password)
```

方法4: 期間を指定して特定の人の変更

```
# Tanakaさんの今日の変更
git log --author="Tanaka" --since="today"

# 昨日から今日まで
git log --author="Tanaka" --since="yesterday"

# 1週間以内
git log --author="Tanaka" --since="1 week ago"

# 特定期間
git log --author="Tanaka" --since="2025-10-01" --until="2025-10-07"
```

方法5: 複数人を指定

```
# TanakaさんまたはSatoさん
git log --author="Tanaka\|Sato"

# 読み方: \| = または(OR)
```

方法6: git blameで行ごとに確認

```
# ファイルの各行を誰が書いたか確認
```

git blame login.py

表示:

abc1234 (Tanaka Taro 2025-10-06 14:30:00 +0900 1) def authenticate(username,
password):
abc1234 (Tanaka Taro 2025-10-06 14:30:00 +0900 2)   user = User.find(username)
def5678 (Sato Hanako 2025-10-05 10:00:00 +0900 3)   if user:
def5678 (Sato Hanako 2025-10-05 10:00:00 +0900 4)   return
user.check_password(password)
abc1234 (Tanaka Taro 2025-10-06 14:30:00 +0900 5)   return False

意味:

- 各行の先頭 = コミットID
- 括弧内 = 作者名、日時
- 最後 = 行番号と実際のコード

見やすくする
git blame -L 10,20 login.py
-L 10,20 = 10行目から20行目だけ表示

方法7: VSCodeのGitLensを使う(超便利!)

VSCodeにGitLens拡張機能をインストールすると:

1. コード上にカーソルを置く
2. 行の右側に「〇〇さんが△月△日に変更」と表示される
3. クリックすると詳細が見れる

🎯 第7章: 実践コマンド集

朝のチェックルーチン

1. プロジェクトフォルダへ移動
cd /c/projects/my-project

2. 現在の状態確認
git status

3. リモートの最新情報取得
git fetch origin

```
# 4. 自分のブランチ確認
git branch

# 5. mainとの差分確認
git diff --stat origin/main

# 6. 他の人の最新変更を確認
git log origin/main --since="yesterday" --oneline
```

プッシュ前の確認

```
# 1. 変更したファイル確認
git status

# 2. 自分の変更を確認
git diff

# 3. リモートとの差分
git fetch origin
git diff origin/feature/my-work

# 4. コミット履歴確認
git log --oneline -5
```

コンフリクト発生時

```
# 1. どこでコンフリクト?
git status

# 2. 差分確認
git diff login.py

# 3. VSCodeで開く
code login.py

# 4. 解決後、確認
git status
git diff --staged
```

他の人の作業確認

```
# 1. Satoさんの今週の作業
git log --author="Sato" --since="1 week ago" --oneline

# 2. 変更されたファイル
git log --author="Sato" --since="1 week ago" --name-only
```

```
# 3. 詳しい差分
git log --author="Sato" --since="1 week ago" -p

# 4. 特定ファイルの履歴
git log --author="Sato" -- login.py
```

🔧 第8章: トラブルシューティング

Q: Git Bashが文字化けする

```
# 日本語設定
git config --global core.quotepath false
git config --global gui.encoding utf-8
```

Q: 差分が表示されない

```
# 確認1: 本当に変更した?
git status

# 確認2: addしちゃった?
git diff --staged

# 確認3: コミットしちゃった?
git log -1 -p
```

Q: リモートとの差分が見れない

```
# まずfetch!
git fetch origin

# それから差分確認
git diff origin/main
```

Q: 特定の人が見つからない

```
# コミッターリスト確認
git log --format='%aN' | sort -u

# 表示例:
Sato Hanako
Suzuki Ichiro
Tanaka Taro
```

```
# この名前で検索
git log --author="Tanaka Taro"
```

📋 第9章: 完全チートシート

Git Bash起動

```
# フォルダで右クリック → Git Bash Here
# または VSCode → Terminal → Git Bash
```

状態確認

```
git status              # 現在の状態
git branch              # ブランチ一覧
git branch -a           # リモート含む全ブランチ
git remote -v           # リモートURL
git log --oneline -10   # コミット履歴
```

差分確認

```
git diff                    # 作業中の変更
git diff --staged           # addした変更
git diff origin/main        # リモートとの差分
git diff abc123 def456      # コミット間差分
git diff --stat             # 統計表示
```

コンフリクト

```
git status              # コンフリクト箇所確認
git diff ファイル名      # 差分詳細
code ファイル名          # VSCodeで開く
git add ファイル名       # 解決後
git commit              # コミット
```

特定の人の変更

```
git log --author="名前"                  # コミット履歴
git log --author="名前" --since="1 week"   # 期間指定
git log --author="名前" -p               # 差分付き
git blame ファイル名                     # 行ごとの作者
```

💡 重要ポイント

1. 必ず作業フォルダで **Git Bash** を起動
2. **git fetch** してから差分確認
3. **VSCode** の差分表示が見やすい
4. コンフリクトは **VSCode** で解決が楽
5. わからなくなったら **git status**

これで一通りの操作ができるようになりました!

他に知りたいことはありますか?例えば:

- 「プルリクエストの作り方」
- 「ブランチの削除方法」
- 「間違えてコミットした時の対処」
- 「.gitignoreの書き方」

具体的な状況があれば教えてください!