Git 高级操作与原理

このブログ記事は ChatGPT-4o の協力を得て整理されました。Keynote は https://github.com/lzwjava/Keynotes でご覧いただけます。

取り消し

```
$ git commit --amend
$ git add *
$ git status
$ git reset HEAD CONTRIBUTING.md
リセット後の未ステージングされた変更:
M CONTRIBUTING.md
$ git status
$ git checkout -- CONTRIBUTING.md
```

- 再コミット
- ステージングされたファイルを取り消す
- ファイルへの変更を取り消す

コマンド

- `git revert`：ロールバックを行い、新しいコミットを作成して、以前の特定のコミットと相殺します。
- `git cherry-pick`：複数のコミットから特定のコミットを選択します。
- `git rebase`：ブランチのベースを変更し、コミット履歴を整理します。

filter-branch

- 各コミットから特定のファイルを削除する
- ファイルを削除したいが、履歴には残しておきたい
- すべてのコミットからファイルを削除する方法は？

```
$ git filter-branch --tree-filter 'rm -f passwords.txt' HEAD
Rewrite 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)
Ref 'refs/heads/master' が書き換えられました
```

検索

```
$ git grep -n gmtime_r
compat/gmtime.c:3:#undef gmtime_r
compat/gmtime.c:8:return git_gmtime_r(timep, &result);
compat/gmtime.c:11:struct tm *git_gmtime_r(const time_t *timep, struct tm *result)
compat/gmtime.c:16:ret = gmtime_r(timep, result);
compat/mingw.c:606:struct tm *gmtime_r(const time_t *timep, struct tm *result)
compat/mingw.h:162:struct tm *gmtime_r(const time_t *timep, struct tm *result);
date.c:429:if (gmtime_r(&now, &now_tm))
date.c:492:if (gmtime_r(&time, tm)) {
git-compat-util.h:721:struct tm *git_gmtime_r(const time_t *, struct tm *);
git-compat-util.h:723:#define gmtime_r git_gmtime_r
$ git log -SZLIB_BUF_MAX --oneline
e01503b zlib: 一度に4GB以上を供給できるようにする
ef49a7a zlib: zlibは一度に4GBまでしか処理できない
```

.Git ディレクトリ

- config：プロジェクトの設定
- info：.gitignore ファイル
- objects：Git データベース内のすべての内容
- refs：ブランチのポインタ
- HEAD：現在のブランチのポインタ
- index：ステージングエリアの情報

Git オブジェクト

```
$ git init test
空のGitリポジトリが/tmp/test/.git/に初期化されました
$ cd test
$ find .git/objects
.git/objects
.git/objects/info
.git/objects/pack
$ find .git/objects -type f
```

```
$ echo 'test content' | git hash-object -w --stdin
d670460b4b4aece5915caf5c68d12f560a9fe3e4
$ find .git/objects -type f
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

上記のコードは、Git の内部動作を示す一例です。`echo 'test content'`コマンドで生成された内容を、`git hash-object -w --stdin` コマンドを使って Git のオブジェクトデータベースに保存しています。このコマンドは、指定されたデータの SHA-1 ハッシュを計算し、そのハッシュ値を基にオブジェクトを`.git/objects` ディレクトリに保存します。その後、`find .git/objects -type f` コマンドで、保存されたオブジェクトファイルを確認しています。この例では、`d670460b4b4aece5915caf5c68d12f560a9fe3e4` というハッシュ値に対応するオブジェクトが`.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4` というパスに保存されていることがわかります。

- `hash-object`、データを.git ディレクトリに保存するコマンド
- `-w`、オブジェクトを書き込む。指定しない場合、単にキーを返す
- `--stdin`、標準入力から読み取る
- d670...、40 文字のチェックサム
- `cat-file`、Git オブジェクトを確認するための万能ツール

```
$ git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4
test content
```

このコマンドは､指定されたオブジェクト(この場合は`d670460b4b4aece5915caf5c68d12f560a9fe3e4`)の内容を表示します。出力は `test content` です。

```
$ echo 'version 1’ > test.txt
$ git hash-object -w test.txt
83baae61804e65cc73a7201a7252750c76066a30
$ echo 'version 2' > test.txt
$ git hash-object -w test.txt
1f7a7a472abf3dd9643fd615f6da379c4acb3e3a
$ find .git/objects -type f
.git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
$ git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30 > test.txt
```

```
$ cat test.txt
version 1
$ git cat-file -p 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a > test.txt
$ cat test.txt
version 2
$ git cat-file -t 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a
blob
$ git cat-file -p master^{tree}
100644 blob a906cb2a4a904a152e80877d4088654daad0c859 README
100644 blob 8f94139338f9404f26296befa88755fc2598c289 Rakefile
040000 tree 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0 lib
$ git cat-file -p 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0
100644 blob 47c6340d6459e05787f644c2447d2595f5d3a54b simplegit.rb
Tree Objects
```

上記のコードは、Git のオブジェクトデータベースを操作する一連のコマンドを示しています。以下にその内容を日本語で説明します。

1. `echo 'version 1' > test.txt`
   `test.txt` というファイルに「version 1」という内容を書き込みます。

2. `git hash-object -w test.txt`
   `test.txt` の内容を Git のオブジェクトデータベースに保存し、そのオブジェクトのハッシュ値（`83baae61804e65cc73a7201a7252750c76066a30`）を返します。

3. `echo 'version 2' > test.txt`
   `test.txt` の内容を「version 2」に更新します。

4. `git hash-object -w test.txt`
   更新された `test.txt` の内容を Git のオブジェクトデータベースに保存し、新しいハッシュ値（`1f7a7a472abf3dd9643fd615f6da379c4acb3e3a`）を返します。

5. `find .git/objects -type f`
   `.git/objects` ディレクトリ内にあるすべてのファイル（Git オブジェクト）を表示します。これにより、先ほど保存された 2 つのオブジェクトが確認できます。

6. `git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30 > test.txt`
   ハッシュ値 `83baae61804e65cc73a7201a7252750c76066a30` に対応するオブジェクトの内容を `test.txt` に書き出します。

7. `cat test.txt`
   `test.txt` の内容を表示します。ここでは「version 1」が表示されます。

8. `git cat-file -p 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a > test.txt`
   ハッシュ値 `1f7a7a472abf3dd9643fd615f6da379c4acb3e3a` に対応するオブジェクトの内容を `test.txt` に書き出します。

9. `cat test.txt`
   `test.txt` の内容を表示します。ここでは「version 2」が表示されます。

10. `git cat-file -t 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a`
    ハッシュ値 `1f7a7a472abf3dd9643fd615f6da379c4acb3e3a` に対応するオブジェクトのタイプを表示します。ここでは `blob`（ファイルの内容を表すオブジェクト）であることがわかります。

11. `git cat-file -p master^{tree}`
    `master` ブランチの最新のコミットが指すツリーオブジェクトの内容を表示します。これにより、プロジェクトのルートディレクトリにあるファイルやディレクトリの一覧が表示されます。

12. `git cat-file -p 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0`
    ツリーオブジェクト `99f1a6d12cb4b6f19c8655fca46c3ecf317074e0` の内容を表示します。これにより、`lib` ディレクトリ内のファイル（ここでは `simplegit.rb`）が表示されます。

これらのコマンドは、Git が内部的にどのようにデータを管理しているかを理解するのに役立ちます。

```
$ git update-index --add --cacheinfo 100644 \
83baae61804e65cc73a7201a7252750c76066a30 test.txt
update-index
```

このコマンドは、Git のインデックスにファイルを追加するためのものです。具体的には、`test.txt` というファイルを指定したモード（100644）とオブジェクト ID（83baae61804e65cc73a7201a7252750c76066a30）でインデックスに追加します。`update-index` は、インデックスを更新するための Git のサブコマンドです。

- `update-index`、ツリーを作成するコマンド
- `--add`、インデックスを作成
- `--cacheinfo`、Git データベースから読み取る

- 100644、ファイルモード、通常ファイルを表す；100755 は実行可能ファイル；120000 はシンボリックリンク
- 83baae、以前の内容、version 1
- \、1 行のコマンドを 2 行に分割

write-tree

```
$ git write-tree
d8329fc1cc938780ffdd9f94e0d364e0ea74f579
$ git cat-file -p d8329fc1cc938780ffdd9f94e0d364e0ea74f579
100644 blob 83baae61804e65cc73a7201a7252750c76066a30 test.txt
$ git cat-file -t d8329fc1cc938780ffdd9f94e0d364e0ea74f579
tree
```

このコードは、Git の内部コマンドを使用してツリーオブジェクトを作成し、その内容を表示するものです。以下に各コマンドの説明を日本語で示します。

1. `git write-tree`:
    - 現在のインデックス（ステージングエリア）の状態を基に、ツリーオブジェクトを作成します。
    - このコマンドを実行すると、作成されたツリーオブジェクトの SHA-1 ハッシュ値が表示されます。この例では、`d8329fc1cc938780ffdd9f94e0d364e0ea74f579` がそのハッシュ値です。
2. `git cat-file -p d8329fc1cc938780ffdd9f94e0d364e0ea74f579`:
    - 指定されたオブジェクト（この場合はツリーオブジェクト）の内容を表示します。
    - この例では、ツリーオブジェクトが参照しているファイルのモード（`100644`）、タイプ（`blob`）、ハッシュ値（`83baae61804e65cc73a7201a7252750c76066a30`）、およびファイル名（`test.txt`）が表示されています。
3. `git cat-file -t d8329fc1cc938780ffdd9f94e0d364e0ea74f579`:
    - 指定されたオブジェクトのタイプを表示します。
    - この例では、`d8329fc1cc938780ffdd9f94e0d364e0ea74f579` が `tree` タイプのオブジェクトであることが確認できます。

これらのコマンドは、Git の内部的な動作を理解するために役立ちます。

- `write-tree`：ステージングエリアの内容をツリーオブジェクトに書き込みます

read-tree

```
$ echo 'new file' > new.txt
$ git update-index test.txt
$ git update-index --add new.txt
$ git write-tree
0155eb4229851634a0f03eb265b69f5a2d56f341
$ git cat-file -p 0155eb4229851634a0f03eb265b69f5a2d56f341
100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt
100644 blob 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a test.txt
$ git read-tree --prefix=bak d8329fc1cc938780ffdd9f94e0d364e0ea74f579
$ git write-tree
3c4e9cd789d88d8d89c1073707c3585e41b0e614
$ git cat-file -p 3c4e9cd789d88d8d89c1073707c3585e41b0e614
040000 tree d8329fc1cc938780ffdd9f94e0d364e0ea74f579 bak
100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt
100644 blob 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a test.txt
```

このコードは、Git の低レベルコマンドを使用して、新しいファイルを作成し、インデックスに追加し、ツリーオブジェクトを作成し、既存のツリーを読み込んで新しいツリーに統合するプロセスを示しています。以下に各コマンドの説明を日本語で示します。

1. `echo 'new file' > new.txt`
   新しいファイル `new.txt` を作成し、その中にテキスト `new file` を書き込みます。

2. `git update-index test.txt`
   既存のファイル `test.txt` をインデックスに追加します。

3. `git update-index --add new.txt`
   新しく作成したファイル `new.txt` をインデックスに追加します。

4. `git write-tree`
   現在のインデックスの状態を基に、新しいツリーオブジェクトを作成し、そのハッシュ値を出力します。この例では `0155eb4229851634a0f03eb265b69f5a2d56f341` が生成されました。

5. `git cat-file -p 0155eb4229851634a0f03eb265b69f5a2d56f341`
   指定されたツリーオブジェクトの内容を表示します。このツリーには `new.txt` と `test.txt` の 2 つのファイルが含まれています。

6. `git read-tree --prefix=bak d8329fc1cc938780ffdd9f94e0d364e0ea74f579`
   既存のツリーオブジェクト `d8329fc1cc938780ffdd9f94e0d364e0ea74f579` を読み込み、`bak` というディレクトリに配置します。

7. `git write-tree`
   新しいツリーオブジェクトを作成し、そのハッシュ値を出力します。この例では `3c4e9cd789d88d8d89c1073707c3585e41b0e614` が生成されました。

8. `git cat-file -p 3c4e9cd789d88d8d89c1073707c3585e41b0e614`
   新しいツリーオブジェクトの内容を表示します。このツリーには `bak` ディレクトリ、`new.txt`、および `test.txt` が含まれています。

このプロセスを通じて、Git が内部的にどのようにファイルやディレクトリを管理しているかを理解することができます。

コミットオブジェクト

```
$ echo '初めてのコミット' | git commit-tree d8329f
d5ffe1aa4b7b089eee03dccea5e0439ad6d72037
$ git cat-file -p d5ffe1aa4b7b089eee03dccea5e0439ad6d72037
tree d8329fc1cc938780ffdd9f94
```

e0d364e0ea74f579 author lzwjava lzwjava@gmail.com 1462090215 +0800 committer lzwjava lzwjava@gmail.com 1462090215 +0800 最初のコミット

- ハッシュ値は異なります。なぜなら、作成時間と作成者が異なるためです。

$ echo ‘second commit’ | git commit-tree 0155eb -p d5ffe1aa4 e946d6367f07de45cac242dca7cd002f5eaa72b1
$ echo ‘third commit’ | git commit-tree 3c4e9c -p e946d6 09490bf051c34b3693dfd5c7fb63dfe27b295904
$ git log –stat 09490 commit 09490bf051c34b3693dfd5c7fb63dfe27b295904 Author: lzwjava lzwjava@gmail.com Date: Sun May 1 16:38:55 2016 +0800 3 番目のコミット bak/test.txt | 1 + 1 file changed, 1 insertion(+) commit e946d6367f07de45cac242dca7cd002f5eaa72b1 Author: lzwjava lzwjava@gmail.com Date: Sun May 1 16:37:01 2016 +0800 2 番目のコミット new.txt | 1 + test.txt | 2 +- 2 files changed, 2 insertions(+), 1 deletion(-) commit d5ffe1aa4b7b089eee03dccea5e0439ad6d72037 Author: lzwjava lzwjava@gmail.com Date: Sun May 1 16:10:15 2016 +0800 最初のコミット test.txt | 1 + 1 file changed, 1 insertion(+)

- `commit-tree`、最初のパラメータは `blob` または `tree` を指します
- `-p` は親ノードを指します

3 番目のコミット 2 番目のコミット 1 番目のコミット ツリー ツリー ツリー “バージョン 2” “新しいファイル” “バージョン 1” 09490b e946d6 d5ffe1 d8329f 0155eb 3c4e9c 83baae fa49b0 1f7a7a test.txt new.txt test.txt new.txt test.txt バックアップ

ブランチの原理

$ find .git/refs
.git/refs .git/refs/heads .git/refs/tags $ find .git/refs -type f
$ echo “09490bf051c34b3693dfd5c7fb63dfe27b295904” > .git/refs/heads/master $ git log –pretty=oneline
master 09490bf051c34b3693dfd5c7fb63dfe27b295904 3 番目のコミット e946d6367f07de45cac242dca7cd002f5eaa72
2 番目のコミット d5ffe1aa4b7b089eee03dccea5e0439ad6d72037 最初のコミット $ git update-ref refs/heads/master 09490bf051c34b3693dfd5c7fb63dfe27b295904 $ git update-ref refs/heads/test e946d6367f07de45cac242dca7cd002f5eaa72b1 $ git branch * master test

- ポインタ参照を保存する
- `update-ref`、参照を変更または追加するために使用

3 番目のコミット 2 番目のコミット 最初のコミット ツリー ツリー ツリー “バージョン 2” “新しいファイル” “バージョン 1” 09490b e946d6 d5ffe1 d8329f 0155eb 3c4e9c 83baae fa49b0 1f7a7a test.txt new.txt test.txt new.txt test.txt bak refs/heads/master refs/heads/test

$ cat .git/HEAD ref: refs/heads/master $ git symbolic-ref HEAD refs/heads/master $ git symbolic-ref HEAD refs/heads/test $ cat .git/HEAD ref: refs/heads/test

上記のコードは、Gitのシンボリック参照（symbolic reference）を操作する例です。以下に日本語で説明します。

1. `.git/HEAD` ファイルの内容を表示します。現在のブランチが `master` であることがわかります。
2. `git symbolic-ref HEAD` コマンドを使用して、現在のHEADが指している参照を表示します。これも `master` ブランチ
3. `git symbolic-ref HEAD refs/heads/test` コマンドを使用して、HEADを `test` ブランチに切り替えます。
4. 再度 `.git/HEAD` ファイルの内容を表示します。HEADが `test` ブランチを指していることが確認できます。

このように、`git symbolic-ref` コマンドを使用すると、HEADが指すブランチを簡単に変更することができます。

symbolic-ref

タグの原理

$ git update-ref refs/tags/v1.0 e946d6367f07de45cac242dca7cd002f5eaa72b1 $ git tag v1.0 $ git tag -a v1.1 e946d6367f07de45cac242dca7cd002f5eaa72b1 -m ‘テストタグ’ $ cat .git/refs/tags/v1.1 2766532f03289bc5e158629a8b3faffa5f80b8b6 $ git cat-file -p 276653 object e946d6367f07de45cac242dca7cd002f5eaa type commit tag v1.1 tagger lzwjava lzwjava@gmail.com 1462103203 +0800 テストタグ

remotes

$ git remote add origin git@github.com:schacon/simplegit-progit.git $ git push origin master オブジェクトを数える: 11, 完了. オブジェクトを圧縮する: 100% (5/5), 完了. オブジェクトを書き込む: 100% (7/7), 716 バイト, 完了. 合計 7 (差分 2), 再利用 4 (差分 1) To git@github.com:schacon/simplegit-progit.git a11bef0..ca82a6d master -> master $ cat .git/refs/remotes/origin/master ca82a6dff817ec66f44342007202690a93763949

- refs/remotes ディレクトリに配置される
- Remotes の参照とブランチの違いは、それらが読み取り専用であること
- `git checkout` は可能だが、HEAD は変わらないため、commit では remotes の参照を変更できない

$ curl https://raw.githubusercontent.com/mojombo/grit/master/lib/grit/repo.rb > repo.rb $ git checkout master $ git add repo.rb $ git commit -m ‘added repo.rb’ [master 484a592] added repo.rb 3 files changed, 709 insertions(+), 2 deletions(-) delete mode 100644 bak/test.txt create mode 100644 repo.rb rewrite test.txt (100%) $ git cat-file -p master^{tree} 100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt 100644 blob 033b4468fa6b2a9547a70d88d1bbe8bf3f9ed0d5 repo.rb 100644 blob e3f094f522629ae358806b17d test.txt $ git cat-file -s 033b4468fa6b2a9547a70d88d1bbe8bf3f9ed0d5 22044 $ echo ‘# testing’ » repo.rb $ git commit -am ‘modified repo a bit’ [master 2431da6] modified repo.rb a bit 1 file changed, 1 insertion(+) $ git cat-file -p master^{tree} 100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt 100644 blob b042a60ef7dff760008df33cee372b945b6e884e repo.rb 100644 blob e3f094f522629ae358806b17da test.txt $ git cat-file -s b042a60ef7dff760008df33cee372b945b6e884e 22054 $ git gc Counting objects: 18, done. Delta compression using up to 8 threads. Compressing objects: 100% (14/14), done. Writing objects: 100% (18/18), done. Total 18 (delta 3), reused 0 (delta 0) $ find .git/objects -type f $ git verify-pack -v .git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx

この一連のコマンドは、Gitリポジトリ内でファイルを追加、変更、コミットし、その後Gitのガベージコレクション（`git g

1. `curl`コマンドを使用して、`repo.rb`ファイルをリモートリポジトリからダウンロードし、ローカルに保存します。
2. `git checkout master`で、現在のブランチを`master`に切り替えます。
3. `git add repo.rb`で、ダウンロードした`repo.rb`ファイルをステージングエリアに追加します。
4. `git commit -m 'added repo.rb'`で、`repo.rb`ファイルをコミットします。コミットメッセージは「added repo.rb

5. `git cat-file -p master^{tree}`で、`master`ブランチの最新のツリーオブジェクトの内容を表示します。これにより
6. `git cat-file -s 033b4468fa6b2a9547a70d88d1bbe8bf3f9ed0d5`で、`repo.rb`ファイルのサイズを確認します。
7. `echo '# testing' >> repo.rb`で、`repo.rb`ファイルに「# testing」という行を追加します。
8. `git commit -am 'modified repo a bit'`で、変更をコミットします。コミットメッセージは「modified repo.rb a
9. 再度`git cat-file -p master^{tree}`で、変更後のツリーオブジェクトの内容を表示します。
10. `git cat-file -s b042a60ef7dff760008df33cee372b945b6e884e`で、変更後の`repo.rb`ファイルのサイズを確認し
11. `git gc`で、Gitのガベージコレクションを実行し、リポジトリ内の不要なオブジェクトを削除して最適化します。
12. `find .git/objects -type f`で、`.git/objects`ディレクトリ内のファイルをリストアップします。
13. `git verify-pack -v .git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx`で、パックフ

このプロセスを通じて、Gitリポジトリ内のファイルの変更と最適化が行われます。

Refspec

$ git remote add origin https://github.com/schacon/simplegit-progit [remote “origin”] url = https://github.com/schacon/simplegit-progit fetch = +refs/heads/:refs/remotes/origin/ $ git log origin/master $ git log remotes/origin/master $ git log refs/remotes/origin/master fetch = +refs/heads/master:refs/remotes [remote “origin”] url = https://github.com/schacon/simplegit-progit fetch = +refs/heads/master:refs/remotes/origin/ma fetch = +refs/heads/qa/:refs/remotes/origin/qa/ push = refs/heads/master:refs/heads

このコードは、Gitリポジトリのリモート設定と、リモートブランチのログを表示するためのコマンドを示しています。以下に各

1. **リモートリポジトリの追加**:
   ```bash
   $ git remote add origin https://github.com/schacon/simplegit-progit
   ```
   このコマンドは、`origin`という名前でリモートリポジトリを追加します。リポジトリのURLは`https://github.com/schacon/simplegit-progit`です。

2. **リモート設定の確認**:
   ```
   [remote "origin"]
   url = https://github.com/schacon/simplegit-progit
   fetch = +refs/heads/*:refs/remotes/origin/*
   ```
   この設定は、`origin`リモートのURLとフェッチの設定を示しています。`fetch = +refs/heads/*:refs/remotes/origin/*`は、リモートのすべてのブランチをローカルの`refs/remotes/origin/`にフェッチすることを意味します。

3. **リモートブランチのログを表示**:

   ```
   $ git log origin/master
   $ git log remotes/origin/master
   $ git log refs/remotes/origin/master
   ```

   これらのコマンドは、`origin/master` ブランチのコミットログを表示します。`origin/master`、`remotes/origin/master`、`refs/remotes/origin/master` はすべて同じブランチを指しています。

4. **特定のブランチのみをフェッチする設定**:

   ```
   fetch = +refs/heads/master:refs/remotes/origin/master
   ```

   この設定は、リモートの `master` ブランチのみをローカルの `refs/remotes/origin/master` にフェッチすることを指定しています。

5. **複数のブランチをフェッチする設定**:

   ```
   [remote "origin"]
   url = https://github.com/schacon/simplegit-progit
   fetch = +refs/heads/master:refs/remotes/origin/master
   fetch = +refs/heads/qa/*:refs/remotes/origin/qa/*
   ```

   この設定は、`master` ブランチと `qa` ディレクトリ下のすべてのブランチをフェッチすることを指定しています。

6. **プッシュの設定**:

   ```
   push = refs/heads/master:refs/heads
   ```

   この設定は、ローカルの `master` ブランチをリモートの `refs/heads` にプッシュすることを指定しています。

これらのコマンドと設定は、Git リポジトリのリモート操作を管理するために使用されます。

/qa/master

- `<src>:<dst>`
- `refs/remotes` ローカルの位置

https://github.com/schacon/simplegit-progit https://github.com/schacon/simplegit-progit “‘

参考文献

- Pro Git