卒業論文

.gitignore ファイルの実態調査および
リファクタリング手法の提案

08232017　坂本洸亮

指導教員　中丸智貴 助教
森畑明昌 准教授

2025 年 1 月

東京大学教養学部学際科学科総合情報学コース

概要

ここに概要を書く.

目次

第 1 章

はじめに

バージョン管理システムは，システムやソフトウェアの開発において，ソースコードや関連ファイルの変更履歴を記録し，必要に応じて確認・復元するための機能を提供する．その中でも Git は，高速で効率的なバージョン管理システムとして多くのプロジェクトで採用されている．Git は分散型バージョン管理システムと呼ばれ，リポジトリを一つしか持たない集中型バージョン管理システムに対し，複数のリポジトリを運用するのが特徴である．メインサーバに配置された共有リポジトリを開発者が各自で複製し，それに対して独立して作業を行った後，それぞれの変更を統合する．このシステムによって，複数の開発者が並行して作業を行うことが可能となっている．

このようなバージョン管理システムにおいて，管理対象となるファイルを適切に選択することは重要である．ソースコードやビルドファイルなど，変更履歴を管理する必要があるファイルに対して，バイナリファイルやログファイル，ライブラリを配置するディレクトリなどは管理対象外とすることが一般的である．これを実現するために，Git では .gitignore ファイル（以下，単に gitignore と呼ぶ）を用いる．

gitignore は，Git の追跡の対象外とするファイルやディレクトリを指定するもので，リポジトリのルートディレクトリに原則配置される．Git は gitignore の各行で指定されたパターンにマッチするファイルやディレクトリを追跡の対象から除外する．gitignore は，現在追跡されていないファイルやディレクトリが未追跡の状態を保つことを目的とする [1]．そのため，すでに追跡対象となっているファイルやディレクトリについては gitignore で指定するだけでは除外できず，別途 Git のインデックスからファイルを除外する必要がある．

gitignore に記述するパターンとして，除外したいファイルやディレクトリのパスをそのまま利用することもできるが，gitignore が提供する機能を活用すれば，より効率的で柔軟な記述が可能となる．gitignore のパターンで利用できる機能は，メタ文字の形で提供される．メタ文字は，コンピュータプログラムにおいて特別な意味を持つ文字であり，正規表現など文字列を扱うシステムや，プログラミング言語における引用符など，さまざまな分野・文脈で利用されている．それ単体で意味を持つものや，隣接する文字に対して作用するもののほか，二つのメタ文字で文字列を囲むことで複数の文字列にマッチさせることができるものなどが存在

する．

gitignore においては，正規表現で見られるような文字列のパターンマッチングに利用されるメタ文字のほか，ファイルやディレクトリを扱うための特殊なメタ文字を利用することも可能である．例として，ディレクトリの区切り文字を表すスラッシュ（/）や，ディレクトリを再帰的にマッチさせる globstar[*1]（**）などが挙げられる．

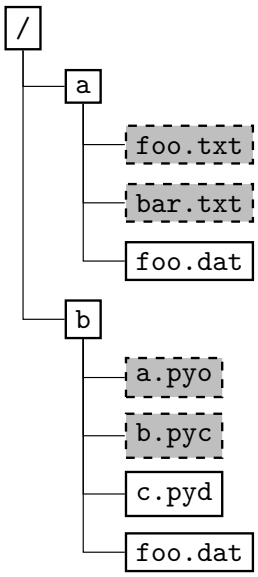

図 1.1　ディレクトリ構造の例

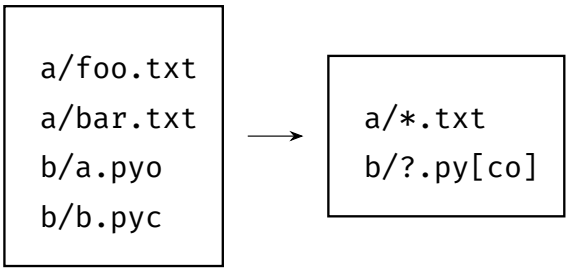

図 1.2　gitignore の記述例

例として図 1.1 のようなディレクトリ構造を考える．破線で示しているものは，gitignore で除外したいファイルであるとする．これを実現するような gitignore の記述例を図 1.2 に示す．図左のように除外したいファイルのパスをそれぞれ記述する代わりに，図右のように gitignore の機能を活用することで，より簡潔な記述が可能となる．

このような記述スタイルは，gitignore の記述を簡潔に保つだけでなく，ファイルやディレクトリのパスを抽象化することで，gitignore の保守性や拡張性の向上にも寄与する．除外対象のファイル名が変更された場合など，gitignore ファイルの修正が必要になった際，それらがパターン化できる限りは既存のパターンの一部を変更するのみでよい可能性が高い．また，図 1.2 の記述に対して，除外する対象として a/baz.txt のようなファイルを追加したい場合，図右のようなアスタリスク（*）を利用した記述により，ディレクトリ a 直下の .txt ファイルをすべてマッチさせることができるため，新しい行を追加する必要がなくなる．このように gitignore の機能を利用することで変更や修正が容易となり，その対象箇所が限定されるため，効率的な保守が可能となるほか，将来的な除外対象の追加に対応しやすくなり，gitignore の拡張性を向上させることができる．

[*1] GNU による Bash Reference Manual [2] での呼称による．

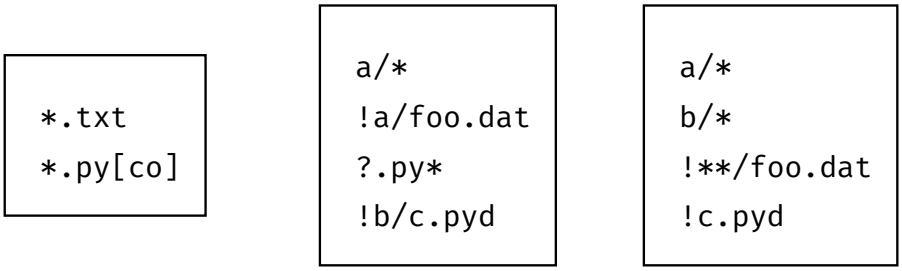

図 1.3　gitignore の記述の多様性

ここで注意すべき点は，機能の利用方法によって同じ結果を与えるパターンが複数存在しうることである．図 1.3 の各記述例は，利用している機能やその利用方法がそれぞれ異なるが，すべて図 1.1 の状態を実現する．このような場合，開発者がそれぞれの環境においてどのような記述が最適であるかを判断するためには，gitignore の性質やその機能の十分な理解が必要となる．

このように，gitignore は単にパスを羅列するだけではなく，その機能を活用して幅広い記述が可能である一方，その柔軟性からかえって記述に冗長性や誤りが生じる可能性もある．ここでいう「冗長性」とは，gitignore の各行について，重複するものが存在したり，ディレクトリ構造的に不要なものが存在することを指す．誤りは実際の gitignore の挙動で発見することが可能だが，冗長性については各パターンをよく観察する必要があり発見が困難である．

また，GitHub などではさまざまな gitignore のテンプレートが提供されており，リポジトリ作成の際にプロジェクトの種類に応じた gitignore テンプレートを選択することができる．簡単に gitignore を作成・利用することができる反面，テンプレートは実際のディレクトリ構造や開発環境など，プロジェクトの状況に適応したものであるとは限らないため，先述したような冗長性が生じやすいと考えられる．

さらに，Visual Studio Code などの統合開発環境（IDE）を利用すると，新しく除外したいファイルを UI から即座に追加することができるなど，gitignore を直感的かつ簡単に利用することができる．しかし，このような機能によって，gitignore のアドホックな変更が容易となるうえ，開発者は gitignore の内容を確認することなく変更を行うことになる。

1.1　本研究の目的

1.2　本研究の貢献

1.3　関連研究

第 2 章

gitignore

2.1　gitignore の各機能の利用状況

2.1.1　仮説

2.1.2　検証方法

2.2　gitignore の記述の冗長性・アドホック性

2.2.1　仮説

2.2.2　検証方法

2.3　gitignore とその他の ignore システム

2.3.1　ignore システムの概要

2.3.2　ignore システムで利用されるパターンマッチングシステム

2.3.3　gitignore とその他の ignore システム間での意味論の比較

第 3 章

gitignore の各機能の利用状況の調査

3.1　データセットの構築

3.1.1　データの収集

3.1.2　復元処理

3.2　定量分析

3.2.1　手法

3.2.2　結果

第 4 章

gitignore の
リファクタリングアルゴリズム

第 5 章

考察

第 6 章

おわりに

参考文献

[1] Git - gitignore Documentation, https://git-scm.com/docs/gitignore.

[2] Bash Reference Manual, https://www.gnu.org/software/bash/manual/bash.html.