|  |
| --- |
|  |
| Git及びGitHub利用マニュアル |
|  |
|  |
| **2015年8月5日　版** |

目次

**はじめに** 2

[**●1.** **Gitによる文書の変更履歴管理** 3](#_Toc426389994)

[**●1.1** **バージョン管理システムを使わない文書の変更履歴管理の問題点** 3](#_Toc426389995)

[**●1.2** **バージョン管理システムを使った文書の変更履歴管理の利点** 4](#_Toc426389996)

[**●1.3** **リモートリポジトリとローカルリポジトリ** 6](#_Toc426389997)

[**●1.4** **文書の変更履歴管理に必要なGitでの操作** 7](#_Toc426389998)

[**●1.5** **Git操作イメージ** 8](#_Toc426389999)

[**●1.6** **まとめ** 9](#_Toc426390000)

**参考文献** 13

**はじめに**

　コンピュータ上で行うプログラム開発や、長期に渡り改定を加えながら使用する文書（条例、マニュアル等）を取り扱う上で、適切な版数管理（履歴管理、変更内容管理、差分管理等）を行うためには、バージョン管理システムを導入する必要があります。

　当マニュアルは、バージョン管理システムのうち、プログラム開発の現場で数多くの利用実績がある分散型バージョン管理システム”Git”、”GitHub”を導入・使用することを提唱し、バージョン管理システムの導入、使用経験がない方、または導入、使用経験はあるが、”Git”、”GitHub”を初めて使用する方にもストレスなく取り扱えるよう、その使い方を解説するものです。

**バージョン管理システムとは**

バージョン管理とは、資源（プログラムや文書）に対して、「誰が」、「いつ」、「何を変更したか」といった情報を記録することで、過去のある時点の状態を復元したり、変更内容の差分を表示できるシステムの事です。これは”版数管理”とも呼ばれます。

**Gitとは**

GitはLinux（Windowsと並ぶオペレーティングシステム）の開発時にあわせて、Linuxのソースコードを効果的に管理するために開発された、比較的新しいバージョン管理システムです。

Linuxの開発では非常に多くのプログラムファイルを扱うため、正確かつ高速にファイルの変更履歴管理が行えるよう工夫されています。

**GitHubとは**

Gitの仕組みを利用して、世界中の人々が自分の作品(プログラムコードやデザインデータ等)を保存、公開することができるようにしたWebサービスの名称です。

　GitHubはGitHub社が運営しており、個人・企業問わず無料で利用することができます。GitHubに作成された資産の格納場所は、基本的に全て公開されますが、有料サービスを利用すると、指定したユーザーからしかアクセスができない、プライベートな資産の格納場所を作ったりすることができます。

# **Gitによる文書の変更履歴管理**

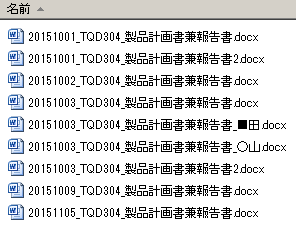
## **バージョン管理システムを使わない文書の変更履歴管理の問題点**

法律の条文や会社の規定など、変更を加えながら長期間利用する電子文書の変更履歴を管理する最もシンプルな方法のひとつとして”**ファイル名による管理**”があります。

この方法は、編集を加える前にファイルをコピーし、ファイル名に日付や編集者の名前等を付与して、いつ、誰が編集したファイルであるかをわかるようにしています。（下記図1.1-1参照）

図1.1-1

日付、編集者名を付与してファイルの管理を行っているが…



しかしこの方法では、ファイル名の付与規則を組織で共有し順守しなければならない。また、変更箇所を確認するには、どのファイルが変更前のファイルかを調べ、変更後のファイルと目視で比較しなければならないため、誤りが発生する可能性が高く、大きな労力を伴う作業が必要となります。

また、組織内で複数名が同時に編集を行い、先に編集した人の変更内容が消える（反映されない）というリスクも発生します。（下記図1.1-2参照）

変更内容を上書きしてしまう可能性もあるし、その確認、修正も難しい…

図1.1-2

製品計画書兼報告書



競合発生

Bさん

Aさん

変更内容A

変更内容B

## **バージョン管理システムを使った文書の変更履歴管理の利点**

バージョン管理システムは、電子ファイル変更履歴管理について前項で示した問題点を解決するために開発されたシステムです。

バージョン管理システムで行える代表的な機能は下記の通りです。

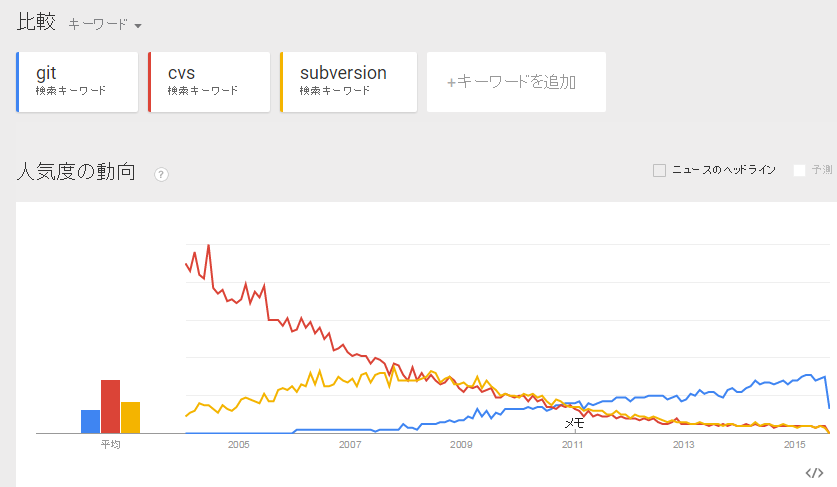
* 最新の原本を取得する
* 変更を原本に反映する
* 変更反映の交通整理をして原本を最新に保つ
* 同時に行われた変更の競合を解決する
* 誤った変更を元に戻す
* 変更履歴の保持

また、バージョン管理システムの代表的な種類は下記の通りです。

* Git　（分散型）
* CVS (Concurrent Versions System)　（集中型）
* Subversion　（集中型）

　上記は現在でも広く利用されている「集中型」、「分散型」と呼ばれる管理システムですが、集中型は資源がメインサーバーに集中するため、サーバーへの接続不可、またはサーバー故障等の場合、資源の取得ができない、または資源消失のリスクがあるため、分散型が主流となっています。

（参考：Googleのトレンド検索結果チャート）



当マニュアルでは、利用価値の高い分散型で、今後更に広く利用されることが見込まれるGitを利用した文書の変更履歴管理を行うための基礎知識を解説します。（下記図1.2-1参照）

　また次章では、WebサービスGitHubを利用して実際に文書の変更履歴管理を行う方法を解説します。

図1.2-1

・原本を手元にコピー

・更新結果の差分を反映

集中型　　　　　　　　　　　　　　　　　　　　**分散型**

原本

原本

原本（写本）

原本（写本）



Bさん



Aさん



Aさん



Bさん

集中型と違い、各々の手元にある原本（写本）を操作するので、複数人で同時に同じ資源への編集が可能。

・原本の取り出し

・更新結果の反映

Ｂさんが作業中のファイルは、Ａさんは編集できない。

## **リモートリポジトリとローカルリポジトリ**

Gitでは、ファイルの格納場所を「リポジトリ」（リポジトリ＝“書棚”のイメージ）と呼び、

2種類のリポジトリを使用して、文書管理を行います。（下記図1.3-1参照）

○**リモートリポジトリ**

クラウド上にある書棚で、現在公開中の最新版文書を格納。この文書は共同で作業しているグループ全体に共有される「原本」となります。

○**ローカルリポジトリ**

あなたが利用している手元のコンピュータ内にある書棚で、「写本（原本のコピー）」を基に行った変更内容を格納します。

図1.3-1

原本

リモートリポジトリ

（共有書棚）



Ｂさん

Ａさん

写本

写本

ローカルリポジトリ

（自分専用書棚）

ローカルリポジトリ

（自分専用書棚）

写本を基に

変更したファイル

写本を基に

変更したファイル



この2段階のリポジトリ（書棚）を連携して利用することで、正確かつ高速な文書管理が行えるよう工夫されているのがGitです。

当マニュアルではリモートリポジトリに原本の文書が既に格納されている状態を起点とし、あなたのコンピュータで文書の変更履歴管理を行うために、必要な知識や操作について説明します。

## **文書の変更履歴管理に必要なGitでの操作**

文書の変更履歴管理を行うために、必要なGitの操作を下記に示します。ここでは文書管理操作に対応するGitの操作には名前があり、代表的なものだけを留意頂ければ構いません。

（下記図1.4-1参照）

|  |  |
| --- | --- |
| 文書管理操作 | Gitでの操作名称 |
| 最新の原本を取得する（ファイルをコピー） | クローン |
| 最新の原本を取得する（更新差分のみ取得） | プル |
| 改変を原本に反映する（リモートリポジトリ宛） | プッシュ |
| 改変を原本に反映する（ローカルリポジトリ宛） | コミット |
| 同時に行われた改変の競合を解決する | マージ |
| 変更履歴の確認 | ログ |
| 変更差分の確認 | ディフ |

図1.4-1

変更履歴・差分

リモートリポジトリ

原本

クローン

プル

コミット

新しいファイル

写本を基に変更したファイル②

ログ

ディフ

マージ

写本を基に変更したファイル①

プッシュ

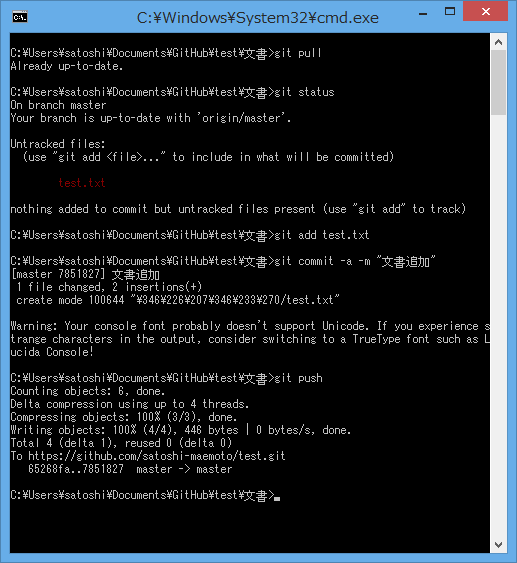
写本

ローカルリポジトリ

## **Git操作イメージ**

Gitはプログラム開発で利用することを目的に作られたツールのため、元来はコマンドプロンプト（コマンドと呼ばれる命令文を用いてWindowsの操作や設定を行うためのツール）で利用します。（下記図1.5-1参照）

図1.5-1



例えば変更をローカルリポジトリに反映するには、命令文の中に下記のように入力しています。

git commit -a -m "文書追加"

しかし、このようなコマンドプロンプトを利用した操作は、覚えなければならないコマンドが多くあり、プログラム開発者ではない方にとって使いやすいとは言えません。

そこでGitには画面を使って操作を行える、Gitを補助するGUIツール（画面上のボタンを押す等の操作を行う事により、コマンド入力を不要にするもの）がいくつか公開されています。GUIツールを利用するとコマンドを覚えなくてもGitを利用でき、変更履歴の差分確認なども視覚的に行うことが可能です。

Windows環境で利用できる代表的なGUIツールとして下記のようなものがあります。

|  |  |
| --- | --- |
| ツール名 | 説明 |
| Tortoise Git　（トータスGit） | Git for Windowsを拡張してWindowsエクスプローラーと統合して利用できるようにしたツール。  Office文書等の差分表示も可能。 |
| Git for Windows | GitをGUIで使えるようにしたもの。 |
| GitHub for Windows | GitHubでプログラム開発を行うケースに特化したツール。 Office文書の差分表示等はできない。 |

本マニュアルではこの中から「Tortoise Git」をピックアップして解説を行います。

　インストール手順は、別紙「Git及びGitHub利用マニュアル（操作マニュアル編）」の

「●1.Tortoise Gitのセットアップ」を参照願います。

## **まとめ**

以上でGitを使用するための基礎知識の解説と、GUIツールを使用する準備を行うことができました。

次章では、GUIツールとWebサービス「GitHub」を利用した文書変更のフローを解説します。

なお、フローの解説については、以下のサンプル文書を例題として用います。この文書は、ある会社で実際に使用されている「品質管理マニュアル」の一部です。

複数回の改版が行われているこのマニュアルを例題として解説することで、Gitを用いたバージョン管理が適切な管理手法であることを示すものとなります。

同文書に変更が必要となり、変更履歴管理を行いながら変更作業を進めるというストーリーで後続の章にて解説を行います。

# 17.不適合製品の管理

[JIS Q 9001:2008　8.3　不適合製品の管理]

当社は不適合製品の識別、処置を行うために、以下の手順に従い不適合製品の管理を行う。

## 17.1　不適合製品の取扱手順

不適合製品の識別は、以下のとおり行う。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | 不適合の適用範囲 | 適用製品  ・サービス | 不適合製品の識別方法 | 不適合製品の  処理手順 |
| １ | 外注品の検証時に検出された不適合 | 受託開発  ソリューション販売 | 外注品検査の結果、“**不合格**”製品をTQR028\_外注品検証記録で識別する。 | 1.PLは、検査結果を協力会社へ  報告する。  2.不合格製品の再納品分について、  再度外注品検査を実施する。 |
| 2 | システムテストで検出された不適合 | 受託開発  ソリューション販売 | TQR042\_不適合製品（苦情）報告書で対象IDを明記する。 | 1.PLは不具合の箇所を究明し、  対象作業工程から修正を行う。  2.修正対象箇所は、再度システム  テストを実施、妥当性確認を行う。 |
| 3 | 製品引渡し後に検出された不適合 | 受託開発  ソリューション販売 | TQR042\_不適合製品（苦情）報告書で対象IDを明記する。 | 1.主管部長は、お客様に不適合  対象機能の一時利用停止を連絡  する。  2.PLは不具合の箇所を究明し、  対象作業工程から製品の修正を  行う。  3.修正対象箇所は、再度システム  テストを実施、妥当性確認を行う。  4.妥当性確認終了後、お客様環境  のシステム入替を申し入れ、お客様と  入替時期を決めて、システム入替  を実施する。 |
| 4 | お客様からのクレーム | ＳＥＳ | 営業部門で、クレームを指摘された技術者を特定しておく。 | 営業部門が、指摘されたクレーム内容を全部とりまとめ、グループ役員会で報告する。 |

★管理責任者は、不適合の発生が実施プロセス上の問題であると判断した場合、18.是正処置、19.予防処置の手順に則り

　是正処置、または予防処置の実施を発令（文書の発行）すること。

# 18.是正処置

[JIS Q 9001:2008　8.5.1　継続的改善、8.5.2　是正処置]

QMS運用において発生した不適合、及び内部監査で指摘された不適合について、以下の手順に従い速やかに是正処置を実施する。

## 18.1是正処置の検出責任

不適合の内容により、是正処置主管部、及び是正処置主管責任者は次の通りとする。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | 不適合の内容 | 是正処置のインプット | 是正処置主管部 | 是正処置主管責任者 |
| 1 | 外注品の検証時の不適合 | TQR028\_外注品検証記録 | 企画部、開発部 | ＰＬ |
| 2 | システムテストで検出されたバグ | TQR042\_不適合製品（苦情）報告書 | 企画部、開発部 | ＰＬ |
| 3 | 納品後に検出されたバグ | TQR042\_不適合製品（苦情）報告書 | 企画部、開発部 | 主管部長 |
| 5 | お客様からのクレーム | TQR042\_不適合製品（苦情）報告書 | 営業部門 | 主管部長 |
| 6 | QMSの不適合 | TQW311\_是正処置要求書兼報告書 | 管理責任者 | 管理責任者 |

## 18.2是正処置の実施責任

不適合の原因により、是正処置主管責任者は次の通りとする。

|  |  |  |
| --- | --- | --- |
| № | 不適合の原因 | 是正処置実施責任者 |
| 1 | 外注品の問題と判断される場合 | ＰＬ |
| 2 | システム開発プロセス、サービス提供プロセスの問題と判断される場合 | 主管部長 |
| 4 | 技術者に対する顧客からのクレームと判断される場合 | 営業部長 |
| 5 | 当社QMSの運用問題と判断される場合 | 主管部長 |

## 18.3是正処置の実施手順

|  |  |  |
| --- | --- | --- |
| １ | 不適合の原因特定 | 是正処置主管部は発行されたTQW311\_是正処置要求書兼報告書に記載された不適合の内容を確認、原因を特定する。 |
| ２ | 是正処置報告書の発行  （是正処置依頼） | 是正処置主管部長は特定した不適合原因に基づき、是正処置を必要と判断した場合、TQW311\_是正処置要求書兼報告書の“不適合の内容”の欄に記入し、是正処置実施部へ是正処置を依頼する。 |
| ３ | 是正処置の実施 | （１）是正処置実施部は、速やかに不適合の原因を追求し、再発防止を図るための是正処置を決定する。  （２）是正処置実施部は、決定した是正処置を実施する。 |
| ４ | 是正処置の記録～発行 | （１）是正処置実施部は、実施した是正処置の結果をTQW311\_是正処置要求書兼報告書に記録する。  （２）TQW311\_是正処置要求書兼報告書の“不適合の原因“、”是正処置の計画“、”是正処置の結果“欄に記入、是正処置実施責任者の承認後、是正処置主管部（発行元）に回答する。 |
| ５ | 是正処置内容のレビュー | （１）是正処置主管部は、上記是正処置の活動をレビューする。  （２）原因が真の原因で、不適合を再発させないような効果のある是正処置がなされていることを確認し、TQW311\_是正処置要求書兼報告書の“効果の確認･是正処置活動レビュー”欄に記入する。  （３）TQW311\_是正処置要求書兼報告書の是正処置主管責任者承認を得て､是正処置を完了とする。 |

【参考文献】

* + 大塚弘記(2014)　GitHub実践入門　技術評論社　2014年
  + 株式会社ヌーラボ(2012)　サルでもわかるGit入門　<http://www.backlog.jp/git-guide/>
  + 平屋真吾(2013)「ガチで5分で分かる分散型バージョン管理システムGit 」atmarkIT 2013.7.5 <http://www.atmarkit.co.jp/ait/articles/1307/05/news028_3.html>
  + 松島浩道(2009)　Gitを使いこなすための20のコマンド　2009.3.17　<http://osdn.jp/magazine/09/03/16/0831212>