ライフプランのバージョン管理について

目　　　　次

1. 概要
   1. ライフプランについて
   2. 我が家のライフプラン

1.2.1　日常のマネーフローの設計

1.2.2　ライフプランの作成

* 1. バージョン管理の必要性

1.4　Git for Windowsによるバージョン管理

1.4.1　概要

1.4.2　ローカルリポジトリの更新について

1.4.3　ローカルリポジトリの更新と競合につい

　　1.5　TortoiseGitについて

1. TortoiseGitによるバージョン管理環境の構築手順

2.1　準備

2.2　各PCでの共通設定

　　2.3　デスクトップPCでの設定

　　2.4　kazPCおよびnaoPCでの設定

　　 　　2.5　ローカルリポジトリの自動バックアップ

付録１．ローカルリポジトリの削除方法

付録２．git\_ps1エラーの原因と解決策概要

付録３．WinMerge＋xdocdiff Plugin のインストール。

1.1　ライフプランについて

　今も昔も家計簿は日々のお金の使い方を決めるための有効なツールなのであるが、これを続けるのは結構大変で途中で挫折してしまうことが多いように思える。最近では紙につけていた家計簿も、Zaimのような便利なアプリで簡単に管理できるようになってきた。

　ただ最近では社会のグローバル化が進み、為替やそれに伴う物価の変動が家計に与える影響を無視できなくなっている。そして、それに伴い保険や年金などの種類やサービスは複雑化する一方なのでまことに厄介だ。例えば、家を買いたくなった時に今買っても大丈夫なのだろうかとか、毎年２回海外旅行にいっても将来困らないかとか、今の生命保険は十分だろうかあるいは払い続けていけるだろうかなど

を判断するのが難しくなる。過剰に心配して過剰に節約すると家族生活はつまらなくなるだろうし、気前よくやりすぎれば不自由な老後生活を送らなければならない。

　このような危険を減らすためには、日々の暮らしを管理する家計簿だけではなく、長期的視点からの生活設計，つまりライフプランを作成する必要がある。ライフプランが描く資産グラフを判断基準として今この大型出費をしても将来困らないかどうかを判断するわけである。もっとも資産グラフはライフプラン作成時点での推定値なので、その後の物価やサービスの変化に伴って更新しなければならない。それでもライフプランを作成し、更新を続けることで、安心感のある消費生活が得られるだろう。

　後述するように最初のライフプランの作成は結構手間がかかる。過去の預金通帳を分析や日々のマネーフローを整理したり、年金や保険のサービスについてはほぼ無知だったので少し恥ずかしかったが信頼のおける保険会社の担当者に家の資産について全て正直に話して原型を作ってもらったりした。ただ

その後の更新についてはさほど大変ではないと思われるし、定期的に家族で家計状況を点検するのはとてもよいことだと思う。従って、まだ完璧とは言えないながらもライフプランを作ってみてよかったと思っているし、もっと早くつくるべきものだったと考えている。不足なところは気づいたら更新してゆけばよいのだから、気にすることはない。

1.2　我が家のライフプラン

1.2.1　日常のマネーフローの設計

　図1.1は、下記の銀行口座からなる我が家の日常のマネーフローの設計図である。

1. メイン口座 ：　全収入と自動引落しされる基本料金の窓口
2. 共用口座 ：　基本的に年１回上記①より入金し、家庭用途に利用
3. kaz口座、nao口座 ：　基本的に毎月上記②より入金される各々の小遣い口座。
4. 非常時用口座 ：　預金額が計画より大きく下回った時の補填用
5. 実家用口座 ：　現在無人の実家の維持の使用。

QR コード

AI によって生成されたコンテンツは間違っている可能性があります。

図1.1　我が家のマネーフロー（設計図）

　図1.1では②と③の境界がやや不明瞭だが、２名で外出した時の交通費・外食費・娯楽費などは②，

各個人に限定して発生した費用は③としている。このため小遣い額は少なめかもしれないが、非公開で自由に使ってよいとしている。もちろんサブ口座を作ってへそくっても構わない。

1.2.2　ライフプランの作成

　図1.2は図1.1のマネーフロー設計図を基に設計したライフプランである。ライフプランの目的は家庭の資金を適切に確保しながら有効使用するための指針を与えることなので、メイン口座と共用口座の預金額をどのように増減させてゆくべきかを示している。なお上述したようにライフプランは物価やサービスの変化に合わせて更新してゆく必要があるので、短時間で行えるようにExcelで作成している。

ところで図1.1では生活費や保険費，税金などによる収支をどう振り分けるかを決めているので、実績値や契約内容などを調べれば預金額の推移は比較的簡単に計算できる。ただここで忘れてならないことは、今後どんな風に過ごしていきたいか，言い換えると何歳頃にどんな大型収支が見込まれるかをできるだけ具体的にイメージして記入することである。

図1.2のライフプランでは、上部の“生活“ブロックに不動産売却による大型収入，リフォームや冠婚葬祭などでの大型出費，毎年数回の国内旅行費や大型家電購入費など（以下これらの支出をまとめてイベント費と記す）を計上している。これらをうまく実現できれば、良い人生が送れたということになるわけである。だからとても重要なブロックである。



図1.2　図1.1のマネーフローに基づいて設計したライフプラン



図1.3　ライフプランと実績の検証用ページ

　次に図1.3左は図1.2のライフプランから総合（メイン口座＋共用口座、以下では生活資金と記す），メイン口座，共用口座の預金額の推移をグラフ表示したものである。総合口座のグラフを見ると、夫が死亡する予定の2043年に生活資金は800万円まで減少するが、直後に1500万円でマンションを売却してその後の妻の生活資金に充てればよいという計画になっていることがわかる。

　ところで我が家には遺産を渡す子供がなく、いざという時の非常用口座もあるので、夫が死亡するまでに夫婦でもう少しお金を使って楽しみたいと思う。そこで仮に図1.2の赤丸セル（Ｚ13セル）に800と入れて2043年のイベント費を800万円増やしてみると生活資金のグラフは図1.4のように、2023年で０になり、その後の妻の生活資金が減少する。つまり妻がこの程度の生活資金でよいと承諾するなら、2043年までのイベント費を800万円増加させて夫婦で楽しむ計画になる。同じ要領で800万円以外のイベント費増加時のグラフも容易に描けるのでよさそうな値に決めればよい。ただ、800万円を超える増加をすると、生活資金が負になって非常用口座で補填に頼らなければならないので、計画としては避けるべきであろう。

　　

図1.4　2043年のイベント費が800万円増えた場合の総合グラフの変化

以上のようにExcelを用いると容易にライフプラン計画時の調整を行える。

　また運用に入ったら、図1.3右につけた表に毎月確認した預金額を記入して実績グラフを追加してゆける。これにより預金額が計画よりずれてきたらすぐにわかるので、その原因を探ってしばらく倹約したり、またはライフプランを見直したりして、早期解決を図ることが出来る。

　このようにExcelはライフプランの単なる記録だけでなく、計画作成時の調整や運用時の問題解決を効率よく行えるのでとても便利である。

1.3　バージョン管理の必要性

　以上のように、ライフプラン（Excelファイル）は一度作ったら終わりでなく、現実離れしないように常に更新してゆかなければならない。しかもライフプランは、家族全員で理解し更新してゆくべきものなので、下記のことが可能なことが要求される。

1. 過去のライフプランからどんな変更をしているか確認したい。
2. 過去のライフプランに戻したい。
3. 家族の誰もが、いつでも気づいた時にライフプランを編集できるようにしたい。
4. 家族の誰もが、簡単に修得できる操作でバージョン管理を行いたい。

　上記①②には、ライフプランに**Git**というバージョン管理ソフトを適用することで対応する。また、

Excelファイルには編集ミスが混入しやすいので、この点でもバージョン管理の適用は有効である。

なお実際のバージョン管理対象は、上記Excelファイルだけでなく、その関連ファイルやサブフォルダを含む集合で、エクスプローラの一部のような木構造関係があるので**ワークツリー**という。

そしてGitは、ワークツリーのバージョンをいつ誰がどこを編集したのか，最新バージョンはどれになるのか等の履歴情報をリポジトリというフォルダ内に格納し、参照可能にするツールである。

　上記③には、**GitHub**というWebサービスを利用することで対応する。

GitHubにリモートリポジトリを作成し、ネットワークを介して各家族がこれにアクセスしながら分散型のバージョン管理を行うことができる。

　そして上記④には、**TortoiseGit**アプリを適用することで対応する。

TortoiseGitを用いると、複雑なGitコマンドを修得しなくても、多くの人がなじみのあるエクスプローラの右クリックメニューからバージョン管理に必要な操作を行うことができる。

　以下、これらについて説明してゆく。

1.4　Git for Windowsによるバージョン管理

1.4.1　概要

　図1.5は、Git for Windows という無料ソフトでライフプランのバージョン管理を行う構成である。

GitHubには１つの**リモートリポジトリ**が作成されており，これにネットワークを介して３台のクライアントPCが接続されている。リモートリポジトリの.gitフォルダにはワークツリーの更新履歴情報が格納されており、これを用いてバージョン０からバージョンrの任意のワークツリーを復元展開できるようになっている（**チェックアウト**）。一方各クライアントPCには**ローカルリポジトリ**が作成されており、.gitフォルダの他にワークツリー内のファイルやサブフォルダを格納している。

因みに.gitフォルダしかないリポジトリを**ベアリポジトリ**，ファイルやサブフォルダなども含むリポジトリを**非ベアリポジトリ**という。図1.3ではリモートリポジトリはベアにしているのでワークツリーを展開する場所がなく、仮に第３者がアクセスしても編集対象のファイルやサブフォルダを直接見られることがない。そして非ベアなローカルリポジトリを持つクライアントPCでだけワークツリーを展開してファイルやサブフォルダを直接見たり編集したりできるようにしている。

以上のことからライフプランの更新は、まず各クライアントPCでローカルリポジトリを使いながらローカルなバージョン管理を行いながらワークツリーの更新案を作成し、次にそれをリモートリポジトリに適用することで行ってゆくのである。

＊当初リモートリポジトリはOneDriveにおいてもOKというWeb記事を見たので、アカウントを増

　やすのも億劫な気がしたのでChatGPTに相談しながら何とか動かそうと散々苦労したが、安定動作

できず、そのうちこんなことを言ってきた。

✅**“OneDrive, Google Drive, Dropbox などのクラウドストレージは非推奨**

理由：

１．Git は複数のファイルを一斉に管理するが、クラウド同期は「ファイル単位」で処理する

・そのためgit push や git fetch の途中でクラウド同期が動くと、リポジトリが壊れるリスクがある

２．意図しない競合が発生する

　　・複数人が OneDrive 上のリポジトリを使うと「競合ファイル (\_conflicted copy)」が生成される

・これが原因で git status で謎の変更が出たり、プッシュできなくなることがある

３．Git の仕組みとクラウドの仕組みが合わない

　　・Git は objects や refs の内部構造を使ってデータ管理しているが、

クラウドストレージの「差分同期」はこれを無視してしまうことがある

💡 解決策 →

✅ GitHub / GitLab / Bitbucket などのリモートリポジトリサービスを使うのが最善

✅ 自力で Git サーバーを立てるなら、SSH か Git over HTTP(S) を利用する

もっと早くいってよ！と思ったが、やはり餅は餅屋ということでGitにはGitHubがいいんだな。



図1.5　Git for Windows による　ライフプランのバージョン管理構成

1.4.2　ローカルリポジトリの更新について

図1.6は、図1.5のGitバージョン管理の基本的な流れのイメージを表したものである。



図1.6　 Gitバージョン管理の基本的な流れのイメージ例

　図1.6では、まずkazがリモートリポジトリからrkバージョンを**プル**してローカルリポジトリに取込んでワークツリーの編集を行い、ローカルリポジトリにコミットし、さらにリモートリポジトリに**プッシュ**することでrk＋１バージョンに１つ更新される。そして次にnaoがrn（n≧k）バージョンをプルして同様の作業をしている。

つまり、kazとnaoの作業時間（プルからプッシュまでの時間）が重なっていないので、リモートリポジトリの履歴は連続性を保ったまま直線で伸びている。

　なお図1.6からわかるように、各ローカルリポジトリのコミットはできるだけこまめにすべきである。変更誤りやその修正，やり直しといった経過内容まで記録すると履歴が汚くなるのではと心配するかもしれないが、ローカルリポジトリは１作業毎にプルして書換えるので気にすることはない。

ローカルリポジトリの目的は、次のリモートリポジトリ更新案を適切に作ってゆくことなのだから、

１作業時間中はより詳細な履歴情報を記録する方が重要になるからだ。といってもあまりに見にくくなるのも問題だという時には、コミットを指定のバージョンまで戻し（**リセット**）、整理することもできる。

＊図1.15ではリモートリポジトリをローカルリポジトリに取込む方法としてクローンとプルがある。

　　ここでクローンはリモートリポジトリを丸ごとローカルリポジトリにコピーするもので、コピー先のフォルダは空である必要がある。一方プルはリモートリポジトリの最新変更でローカルレポジトリを更新するものだが、編集中のファイルやサブフォルダは不変、という違いがある。

従って後述するように、ローカルリポジトリはクローンで作成し、その後は時々プルを行って

リモートリポジトリと同期を取りながら更新してゆくのがよいとされている。

1.4.3　ローカルリポジトリの更新と競合について

　図1.6のように各作業時間が重ならないように作業が進めばよいが、各家族がいつ作業を始めるかは制約しないので、例えば図1.7のような競合が発生してプッシュできなくなる場合がある。



図1.7　競合が発生する時のイメージ例

　図1.8(1),(2)は、図1.7のような競合を解決する代表的な方法である。どちらを用いるかはその場に応じて決めればよい。



1. **マージ**　による解決



**(2)リベース**　による解決

図1.8　競合を解決する２つの代表的な方法

　なお、図1.8に示したマージもリベースもGitコマンドの入力で容易に行えるようになっているが、

やはり競合の発生頻度が高まると作業効率が低下するので、各クライアントPCでは下記のようにリモートリポジトリをこまめにプル取得しながら作業することが大切である。因みにプルではローカルリポジトリ内に展開し編集中しているワークツリーは上書きされない。

**１．各作業を開始する直前に、まず必ずプルする。**

もし図1.7でnaoが最初のコミット近く（つまりもう少し右側で）でプルできていれば、

naoはrkバージョンでなくrk+1バージョンを編集するので図1.7は図1.6と同じになり、競合は発生しなかったはずである。

つまり最初のコミット直前にプルすれば競合発生頻度を低減できるわけであるが、作業開始から最初のコミットまでにどのくらい時間がかかるかはわからないので、現実的には作業開始直前にプルするということになる。不完全であるが、最も基本的な必須ルールである。

**２．編集中のローカルなワークツリーを、コミットやプッシュする直前にプルする。**

　リモートリポジトリは他のクライアントからプッシュ更新されるので、編集中のワークツリーをいきなりプッシュしても競合してしまうかもしれない。そこでプッシュしたいクライアントPCがプッシュすると競合が発生するかどうかを確認し、発生するなら事前に解決してプッシュしようというものである。

具体的には、まずプルしてローカルリポジトリの.gitフォルダだけをリモートリポジトリのものに置換え、編集中のワークツリーをコミットしてエラーがでるなら競合が発生すると判定するわけだ。そしてその場合は、エラーがなくなるようにワークツリーを修正し（図1.8参照）、コミット，続いてプッシュを行う。

**３．他のクライアントと共同編集しているファイルを編集する直前にプルする。**

　　以上ではワークツリー単位での説明をしてきたが、実はkazとnaoが同じファイルを変更

しないなら図1.7においてプッシュしてもリモートリポジトリの連続性は保たれ、競合は

発生しないことは理解できるだろう。

　つまりワークフローの各ファイルに選任担当者が決まっていたら競合は発生しないので、

複数担当者が割当てられたファイルを編集する時だけ上記１のようにプルして作業する。

ただこの方法は、各ファイルの担当者をしっかり決めておくこと前提になる

　補足）実際のローカルリポジトリの更新は、実験的に更新箇所や更新方法を変えて試したりするので、

必ずしも図1.6で示したような直線的には行われず、内部でさらに分岐したり結合したりする。

この分岐経路は**ブランチ**というが、当然ブランチの結合の際に競合が発生することがある。ただ

その場合は、上述と同様な対応をすればよいだろう。

1.5　TortoiseGitについて

　Git for WindowsをインストールしたPCで、スタートメニュー→[すべてのアプリ]→[Git]→[Git Bash]と入力するとGit Bashコンソールが開くので、下記コマンドを入力してGitについて知ることができる。

$ git help git → gitのマニュアルページが開き、使い方を知ることができる

$ git –help → Git Bashコンソール上にGitコマンドの構造と一覧が表示される

これらからGit BashコンソールにGitコマンドを入力することにより、コミットやプル、プッシュ等の操作を行える上に、フォーク，チェリーピックなどのコマンドも用意されていて、とても多機能なソフトであることがわかる。

　ただGitになじみのない人がこれらのコマンドを修得するには時間がかかり、肝心のバージョン管理をなかなか開始できなくなる可能性がある。そこでTortoiseGitを追加インストールし、図1.9のように

なじみ深いエクスプローラから直感的に操作できるようにする。



図1.9　TortoiseGitを使ったバージョン管理

1. TortoiseGitによるバージョン管理環境の構築手順

　本章では図1.5の構成に基いて、Git＋TortoiseGitでバージョン管理する環境構築手順について下記のように述べてゆく。

2.1　準備

　 【１】全てのクライアントPCにTortousGitをインストールする。

　 【２】差分比較用アプリをインストールする。

　 【３】必要となるパラメータ値の決定

2.2　各PCでの共通設定

【１】Gitユーザー情報の登録（各PCで１回だけ）

【２】TortoiseGitの言語設定（各PC）

2.3　デスクトップPCでの設定

1. GitHubにリモートリポジトリを作成
2. ローカルリポジトリの作成と設定
3. 動作確認（リモートリポジトリへの最初のコミット＆プッシュ）

2.4　kazPCおよびnaoPCでの設定

【１】ローカルリポジトリの作成と設定（リモートリポジトリの**クローン**）

【２】動作確認（リモートリポジトリへの最初のコミット＆プッシュ）

2.5　ローカルリポジトリへの自動バックアップ（任意）

　なお本章の内容は、ChatGPTに聞きながら実際に構築していった際に作成したメモを整理したものなので、都度気づいたコメントを含めた文章が主体の記述になっている。ただ前章までの内容が理解できていれば、読み進めていけると思うので、不足の部分は別途調べて行ってほしい

2.1　準備

　【１】下記URLを参考に、全てのクライアントPCにTortousGitをインストールする。

Git for Windows と TortoiseGit のインストール【改訂版】

<https://qiita.com/mmake/items/63a869272c0dfa1d50a4>

　【２】差分比較用アプリをインストールする。

TortoiseGitに異なるバージョンのワークツリーの比較を行う際に用いるdiffアプリには、

様々な選択が可能であるが、ここではWinMerge＋xdocdiff Pluginを用いる。

　　　　　無料の定番アプリであるWinMergeにxdocdiffを追加すると、テキストファイルだけでなく

Excelファイルも下図のように分かり易く比較できる。詳しくは付録３を参照。



　【３】必要となるパラメータ値の決定

表2.1は図1.5の構成を構築する上で必要となるパラメータ値である。

以下本章では具体的な設定手順を記述してゆくが、青太文字の部分は表2.1を参照する。

表2.1　設定に必要なパラメータ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 設定内容 | パラメータ名 | DesktopPC | kaz\_PC | nao\_PC |
| Git Hubへの  ユーザー情報登録 | **Your Name** | hokahoka | inogai.kazunori | inogai.naomi |
| **Your.email** | [kinogai321@gmail.com](mailto:kinogai321@gmail.com) | [kinogai321@gmail.com](mailto:kinogai321@gmail.com) |  |
| リモートリポジトリ作成 | **リモートリポジトリ名** | LifeplanRepo | リモートリポジトリはデスクトップPCで  作成するので不要とする | |
| **GitHub Mail** | [kinogai321@gmail.com](mailto:kinogai321@gmail.com，pin3590606) |
| **GitHub Password** | pin3590606 |
| **GitHub Username** | kinogai |
| **リモートURL** | https://github.com/hokahoka/LifeplanRepo.git |
| ローカルリポジトリ作成 | **Gitフォルダ** | C:\Users\PC\_USER\Documents　 （共通） | | |
| **ローカルリポジトリ名** | LocalLifeplanRepo　　　　　　　　　　　　　 （共通） | | |
| **ローカルリポジトリフォルダ** | **Gitフォルダ**\LocalLifeplanRepo　　　　　　　 （共通） | | |
| **リモートリポジトリエイリアス** | origin　　　　　　　　　　　　　　　　　　　 （共通） | | |
| リモートリポジトリの  バックアップ | **バックアップフォルダ** | **ローカルリポジトリフォルダ**\GitBackup　　　 （共通） | | |
| **ミラーリポジトリフォルダ** | **バックアップフォルダ**\LocalLifeplanRepo.git （共通） | | |
| **ミラー更新バッチファイル** | **バックアップフォルダ**¥update\_mirror.bat　　 （共通） | | |

2.2　各PCでの共通設定

【１】Gitユーザー情報の登録（各PCで１回だけ）

1. エクスプローラーで適当なフォルダをShift+右クリック → Git Bash Here を開く。
2. 以下のコマンドを入力（各PC毎に設定。青地部分は表2.1参照）

git config --global user.name **"Your Name"**

git config --global user.email **"your.email"**

＊上記で“git\_ps1: command not found”エラーがでても‘git config --global user.name’

　コマンドなどで正しく登録されていることを確認することができる。

1. 下記コマンドの出力内容で設定を確認

git config --global --list

【２】TortoiseGitの言語設定（各PC）

1. エクスプローラーで適当なフォルダを右クリック → TortoiseGit → Settings を開く
2. General（一般）タブ → Language（言語）を「日本語」に変更 → OK

2.3　デスクトップPCでの設定

【１】GitHubにリモートリポジトリを作成

1. GitHubにログイン（[GitHub](https://github.com/" \t "_new)：<https://github.com/>）
2. 右上の「＋」ボタンから「New repository」を選択
3. **リモートリポジトリ名**を入力

(4)「Initialize this repository with a README」はチェックしない

(5)「Create repository」をクリックし、下記を設定

**GitHub Mail, GitHub Password, GitHub Username, リモートURL**

【２】ローカルリポジトリの作成と設定

(1)**ローカルリポジトリフォルダ**を新規作成

(2)ローカルリポジトリを初期化

**・ローカルリポジトリフォルダ**を右クリック → Git ここにリポジトリを作成

・「ベアリポジトリにする」はチェックしない

・OK をクリック

(3)ローカルリポジトリにリモートリポジトリを登録

・**ローカルリポジトリフォルダ**を右クリック → TortoiseGit → リモートの追加

・**リモートリポジトリエイリアス**と**リモートＵＲＬ**を設定する

＊リモートの追加メニューが表示されない時はMyLocalRepoフォルダの右クリックから

Git Bashを開き、下記コマンドで手動追加する。

　　 git remote add **リモートリポジトリエイリアス** "**リモートURL** "

　下記コマンドの応答がorigin/masterなら成功だが、何も表示されなければ失敗。

git remote -v

　例えばパスのエラーなら、下記のように再作成すれば修正できる。

git remote remove **リモートリポジトリエイリアス**

git remote add **リモートリポジトリエイリアス** "**リモートURL** "

【３】動作確認（リモートリポジトリへの最初のコミット＆プッシュ）

1. **ローカルリポジトリフォルダ**内に README.md など適当なファイルを作成
2. **ローカルリポジトリフォルダ** を右クリック → Gitコミット -> “master”
3. README.md にチェックを入れ、コミットメッセージに例えば“最初のコミット”と書く→ コミット
4. 下記のようにリモートリポジトリにプッシュ

・**ローカルリポジトリフォルダ**を右クリック → TortoiseGit → プッシュ

**・リモートリポジトリエイリアス** を選択し、masterブランチをプッシュ→ OK

2.4　kaz\_PCおよびnao\_PCでの設定

【１】ローカルリポジトリの作成と設定（リモートリポジトリの**クローン**）

1. 下記コマンドにて、GitHubからリモートリポジトリをクローンする

cd **Gitフォルダ**

git clone **リモートURL** **ローカルリポジトリ名**

(2)ローカルリポジトリが作成されたことを確認

cd **ローカルリポジトリフォルダ**

ls -a

【２】動作確認（リモートリポジトリへの最初のコミット＆プッシュ）

　　　　　(1)ファイルを変更してコミット

・**ローカルリポジトリフォルダ**内に新しいファイルhello.txtを追加・編集

・**ローカルリポジトリフォルダ**を右クリック → Git コミット -> “master”

・hello.txtにチェックを入れ、コミットメッセージを入力 → OK

　　　　　(2)リモートリポジトリの最新の変更をプルして、プッシュ更新しても競合しないか判定する。

・**ローカルリポジトリフォルダ**を右クリック → TortoiseGit → プル

・リモートの変更を取得してローカルに適用

　　　　　(3)判定結果に応じて、下記のように対応する

・エラーメッセージが出た（競合あり）の場合は、下記のように行う。

　　 　　　マージまたはリベースでローカルリポジトリを修正し、上記(2)に戻る。

・エラーメッセージがでない（競合無し）の場合は、通常のプッシュ

**ローカルリポジトリ**フォルダを右クリック → TortoiseGit → プッシュ

**リモートリポジトリエイリアス**を選択し、**masterブランチ** をプッシュして終了

2.5　リモートリポジトリへの自動バックアップ

　バージョン管理を長期間運用してゆくと、クライアントPCやネットワークにおける不具合や操作ミスなどにより大切なリモートリポジトリが壊れることがあるかもしれない。ただ、ここまでの記述で、このような場合も定期的にリモートリポジトリを自動バックアップしていれば、前の状態に復帰できるので安心であることは理解できるだろう。

　図2.1は、図1.5にデスクトップPCにミラーリポジトリを追加したものである。

ミラーリポジトリにリモートリポジトリを定期的に自動バックアップしておけば、障害発生時にはGitHubに新しいリモートリポジトリを作成してミラーリポジトリをプッシュすることで、最新のバックアップ時点での全履歴情報を復元できる。

さらに他のクライアントPCでも同様のバックアップを行えば、デスクトップPCに不具合が発生しても他のクライアントPCからリモートリポジトリを復元できるのでよい安心できる。

以下ではこれらについて説明してゆく。なお、説明はデスクトップPCを含む全クライアントPCに対しても共通である。



図2.1　リモートリポジトリのバックアップ構成

【１】クライアントPCにミラーリポジトリを作成

　　　　(1)クライアントPCにミラーリポジトリを作成

mkdir**バックアップフォルダ**

(2)ミラーリポジトリの作成

git clone --mirror**リモートURL** **ミラーリポジトリフォルダ**

【２】クライアントPCの**バックアップフォルダ**中に自動更新用バッチファイルを作成

1. 下記内容の**ミラー更新バッチファイル**作成

@echo off

cd /d **ミラーリポジトリフォルダ**

git fetch --all

echo Update completed at %DATE% %TIME% >> **バックアップフォルダ**\log.txt

　【３】クライアントPCのタスクスケジューラで自動実行の設定

1. タスクスケジューラを開く。
2. ［タスクの作成］を選択し、下記タブ設定をする

・［全般］タブ

名前: GitBackup

[ユーザーがログオンしているかどうかにかかわらず実行する] にチェック

管理者権限で実行する

・［トリガー］タブ

[新規] をクリック

毎日、４時間ごとに実行するなど適宜設定

・［操作］タブ

[新規] をクリック

　　[プログラム/スクリプト] に**ミラー更新バッチファイル名** を指定

・［OK］をクリックしてタスクを保存

1. ログ出力は、　git log –omline –all　で確認できる。

付録１．ローカルリポジトリの削除方法

ローカルリポジトリは、「.git」フォルダを削除 することで削除する（通常のフォルダに戻す）ことができる。

ローカルリポジトリの削除手順は、下記の2通りがある。

**方法①：TortoiseGitメニューから削除**

1. **削除したいローカルリポジトリのフォルダを右クリック**
2. TortoiseGit → このリポジトリの管理を削除 をクリック
3. 確認ダイアログが出るので OK を押す

**これで「.git」フォルダが削除され、通常のフォルダに戻る**

**方法②：手動で削除**

1. エクスプローラーでローカルリポジトリのフォルダを開く
2. **隠しフォルダを表示する**（必要なら）
   * 表示 タブ → 隠しファイル をチェック（Windows 10/11の場合）
3. **「.git」フォルダを削除**
4. **フォルダを削除する場合** → そのまま Shift + Delete で完全削除

**これでローカルリポジトリごと削除できる。**

付録２．git\_ps1エラーの原因と解決策

例えばGit BashでGitにglobal user.nameを設定すると、下記のようなエラーが出ることがある。

$ git config --global user.name **"hokahoka"**

sh: \_\_git\_ps1: command not found

メッセージ中の\_\_git\_ps1はGit のプロンプト（PS1）にブランチ名を表示する関数だが、Git for Windows の環境によっては正しく読取れないことがあるので、その場合は下記の解決策を順に試す。

**解決策1：PowerShell / コマンドプロンプトを挑戦する**

Git Bash の代わりにPowerShell やコマンドプロンプト（cmd）で設定できるか試してみる。

1. Windowsキー + R →cmdと入力 → Enter
2. 以下のコマンドを実行：

git config --global user.name "**hokahoka**"

1. 設定が反映されたか確認：

git config --global user.name

これでエラーが出ずに**hokahoka**表示されればOK！

**解決策 2：Git Bash の設定を修正**

Git Bashを使いたい場合、\_\_git\_ps1エラーが出ないようにまたは.bashrcを.bash\_profile修正できる。

1. Git Bash を開く
2. 以下のコマンドを実行：

nano ~/.bashrc

1. \_\_git\_ps1が使えるように以下の行を追加：

source /usr/share/git/completion/git-prompt.sh

1. Ctrl + X→ Y→Enterで保存
2. 以下のコマンドで変更を適用：

source ~/.bashrc

**これでエラーが出なくなるはず！**

**解決策 3：Git for Windows を再インストールする**

上記の方法でも解決しない場合、Git for Windows が壊れている場合は可能性があるので下記を行う。

1. [**Git for Windowsから**](https://gitforwindows.org/)最新バージョンをダウンロード
2. インストール時に**「Git Bash Here」を有効にするオプションを**選択
3. インストール後、PCを再起動

これで\_\_git\_ps1エラーが解消される可能性が高い。

付録３　WinMerge＋xdocdiff Plugin のインストール

WinMerge（無料のファイル比較ツール）に、Excel・Word対応のxdocdiff Pluginを組み合せると、

.xlsx ファイルをシートの内容として比較できる。

以下にこれをTortoiseGitに追加設定する方法を述べる。

1. 下記URLを参考にWinMergeをインストールする。

<https://qiita.com/frozencatpisces/items/8f998720de8f2aaa7e37>

1. xdocdiff Pluginをインストールする。

(1)下記URLからxdocdiffPlugin64をダウンロードし、解凍する。

https://crus.mydns.jp/xdocdiffPlugin64/

1. 必要なファイルの配置

解凍したフォルダ内の以下のファイルを、WinMerge のインストールフォルダにコピー.

・xdoc2txt.exe​

・zlib.dll 注１）

またMergePluginsフォルダ内の amb\_xdocdiffPlugin.dll を、WinMerge のMergePlugins フォルダにコピーする。。

1. WinMerge の設定

WinMerge を起動し、メニューの「プラグイン」→「自動展開」にチェックを入れる。

1. プラグインの有効化

「プラグイン」→「プラグインの設定」を開き、amb\_xdocdiffPlugin.dll にチェックが入っていることを確認する。

1. TortoiseGit に登録

(1)TortoiseGit の設定 → Diff Viewer タブ → External にチェック

(2)パスに WinMergeU.exe を指定し、以下のように引数を設定する。

/e /u /dl:%base /dr:%mine %base %mine

６．.xlsx を右クリック → TortoiseGit → Diff を選ぶと、比較できる。

注１）一部のバージョンの xdocdiffPlugin64.zip には zlib.dll が含まれていないことがある。

　その場合は、下記のように対応する。

① WinMerge に含まれている場合：

WinMerge をすでにインストールしているなら、

WinMerge フォルダ内に zlib1.dll が入っている場合があるので、

それをそのまま使用して OK （zlib1.dll は zlib.dll の別名で実体は同じ）。

② 自分で zlib.dll を用意する方法

[zlib の公式ページ](https://zlib.net/) ではDLL配布はしていないが、下記のいずれかで取得できる。

(1)下記 GitHub リポジトリで Windows 用のビルド済み zlib DLL を取得可能。

GitHub - madler/zlib:  
<https://github.com/madler/zlib> → Releases に Windows 用 DLL あり

(2)下記配布サイト（安定版）

<https://www.winimage.com/zLibDll/>

→ 「zlib1.dll」 をダウンロードし、zlib.dll にリネームして使用可。