目次

[1. 環境構築編 2](#_Toc16710596)

[1-1. 目標 2](#_Toc16710597)

[1-2. Gitのインストール 2](#_Toc16710598)

[1-3. Gitの設定 9](#_Toc16710599)

[1-4. ローカルリポジトリ環境の構築 10](#_Toc16710600)

[2. 実践編 13](#_Toc16710601)

[2-1. 目標 13](#_Toc16710602)

[2-2. ローカルの修正リソースを、リモートリポジトリへ反映する 13](#_Toc16710603)

[2-3. リモートリポジトリの変更点を、ローカルに反映させる 16](#_Toc16710604)

[3. トラブル編 20](#_Toc16710605)

[3-1. 目標 20](#_Toc16710606)

[3-1.5. トラブル時に必要な基礎知識 20](#_Toc16710607)

[3-2. 修正したファイルを、修正前の状態に戻す 21](#_Toc16710608)

[3-3. ステージング環境にあげたファイルを取り消す 22](#_Toc16710609)

[3-4. コミットしたファイルを取り消す 23](#_Toc16710610)

[3-5. pushしたファイルを取り消す 25](#_Toc16710611)

[3-6. pullしたファイルを取り消す 27](#_Toc16710612)

[4. 補足 – コマンドオプション集 28](#_Toc16710613)

[4-1. git diff 28](#_Toc16710614)

[4-1. git log 28](#_Toc16710615)

[4-1. git checkout 28](#_Toc16710616)

[4-1. git add 28](#_Toc16710617)

[4-1. git commit 28](#_Toc16710618)

[4-1. git pull 28](#_Toc16710619)

[4-1. git push 28](#_Toc16710620)

[4-1. git reset 28](#_Toc16710621)

[4-1. git fetch 28](#_Toc16710622)

[4-1. git merge 28](#_Toc16710623)

[4-1. git rebase 29](#_Toc16710624)

[4-1. git ls-remote 29](#_Toc16710625)

[5. 補足 – エクセルの差分確認ツール 30](#_Toc16710626)

[5-1. 目標 30](#_Toc16710627)

[5-2. ツールの導入 30](#_Toc16710628)

1. 環境構築編

1-1. 目標

① Gitのインストールと、個人設定まで完了する。

② Githubからローカルリポジトリを作成することができる。

③ ブランチを作成して、リモートからソースを落とすことができる。

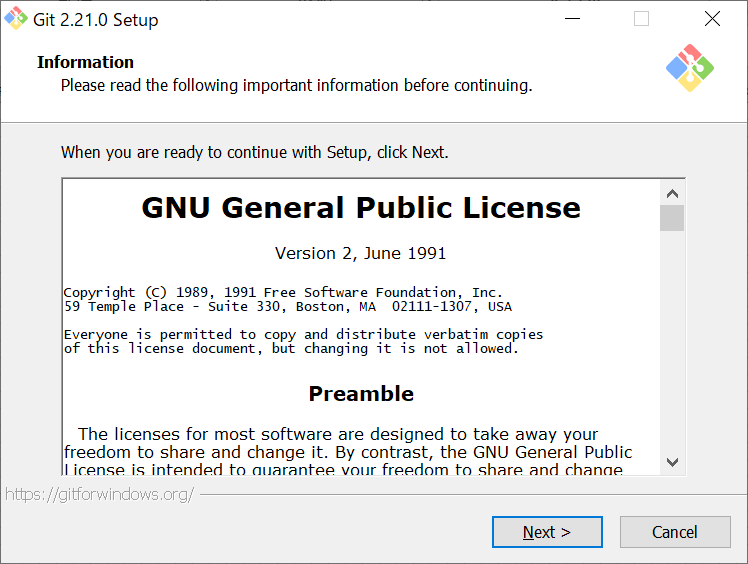
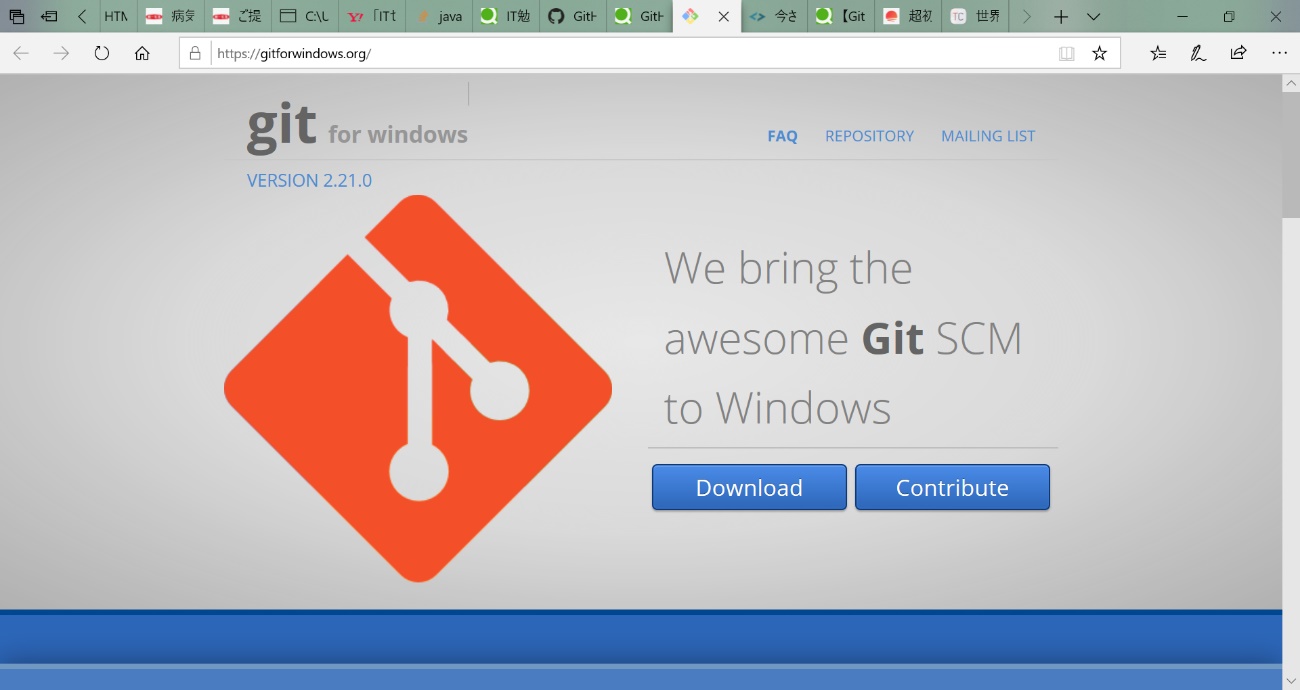
1-2. Gitのインストール

●参考URL：

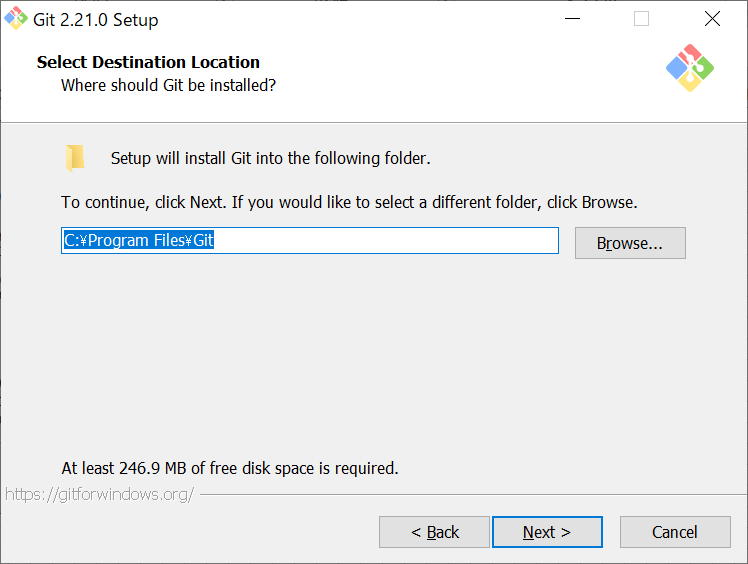
<https://qiita.com/Kenta-Okuda/items/c3dcd60a80a82147e1bf>

1. 以下のURLから、インストーラをダウンロードし、インストーラを実行する

<https://git-for-windows.github.io/>

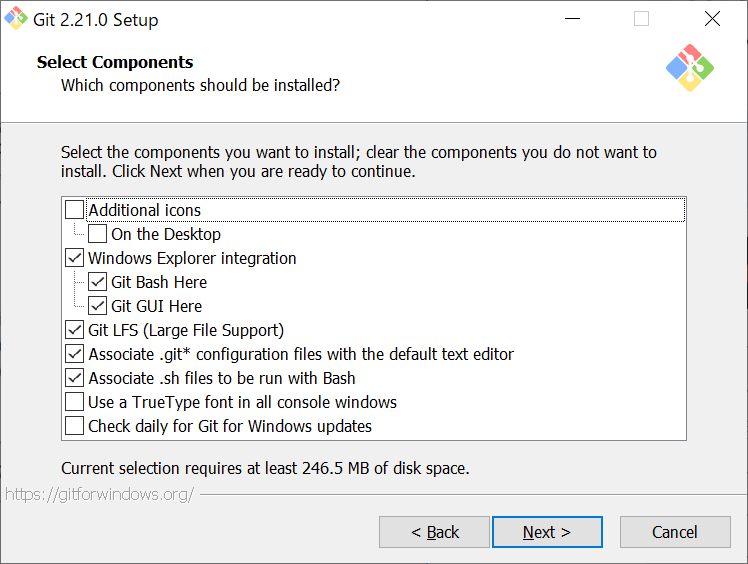


2. インストール先のディレクトリの選択

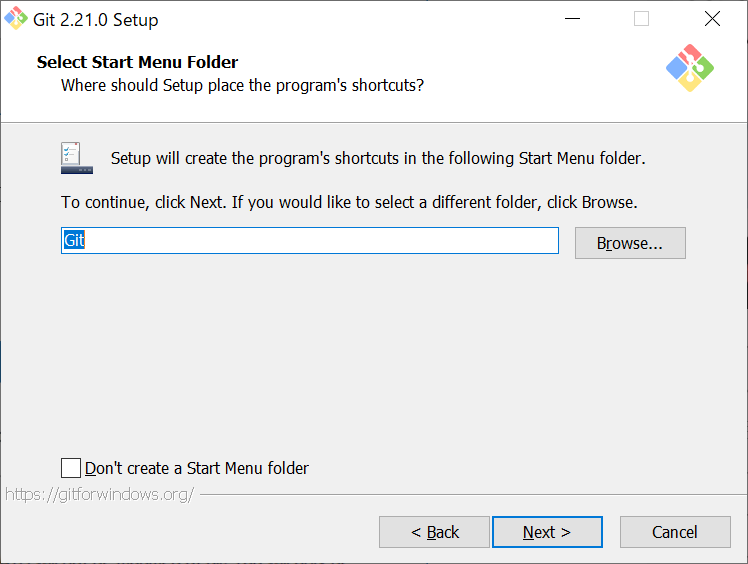


3.コンポーネント導入オプションの選択

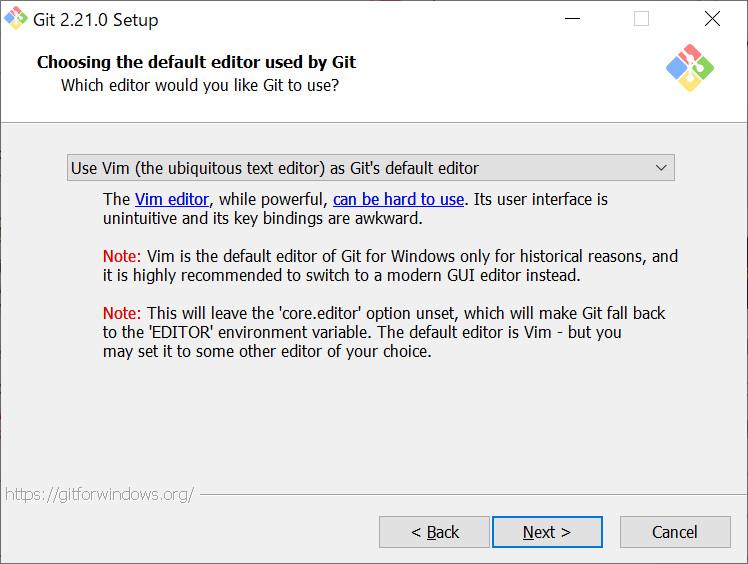
Use a TrueType font in all console windowsには、チェックをいれないように。



4. スタートメニューの追加選択

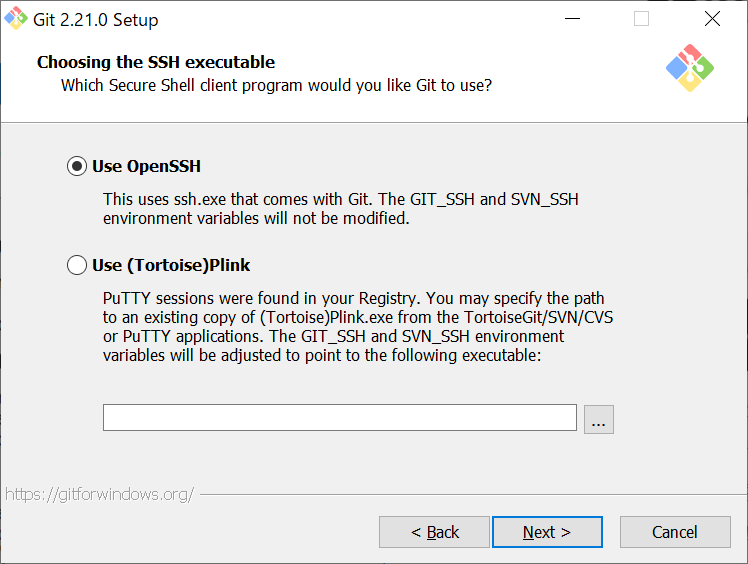


5. Gitで編集するデフォルトのエディタの選択



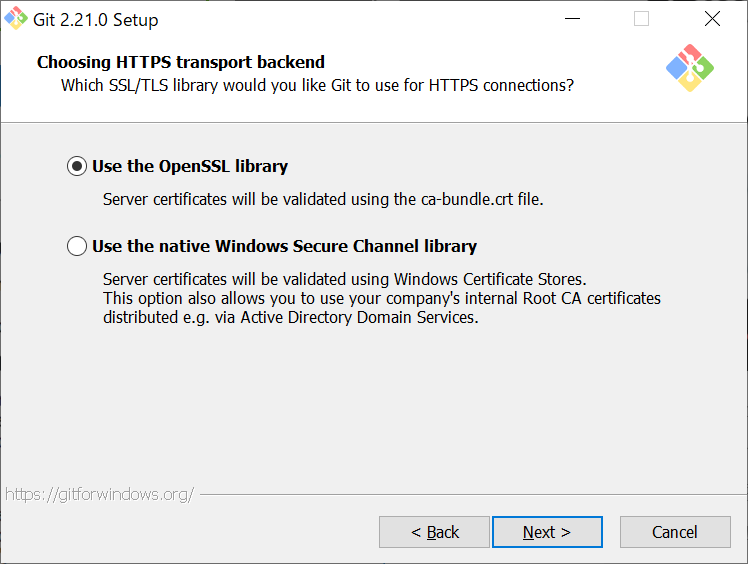
6. SSH接続方式の選択

Puttyをインストールしていたら聞かれる。Gitのsshを使いたいので、UseOpenSSHを選択

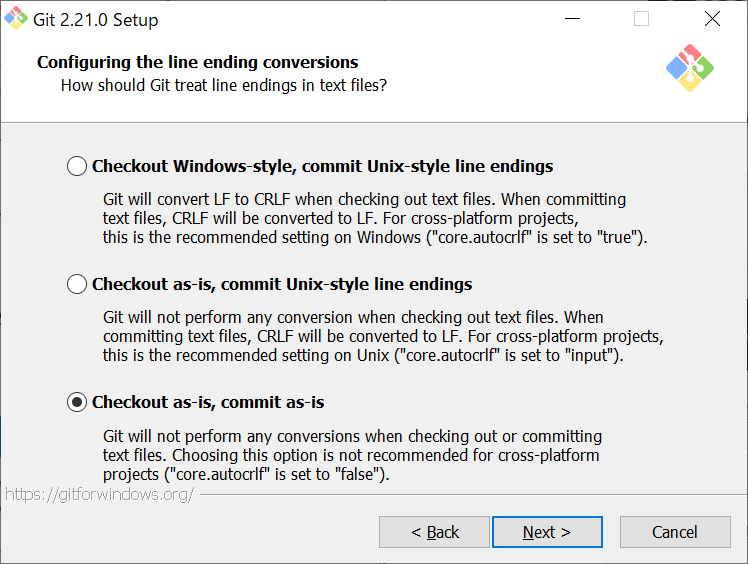


7. HTTPS接続方式の選択

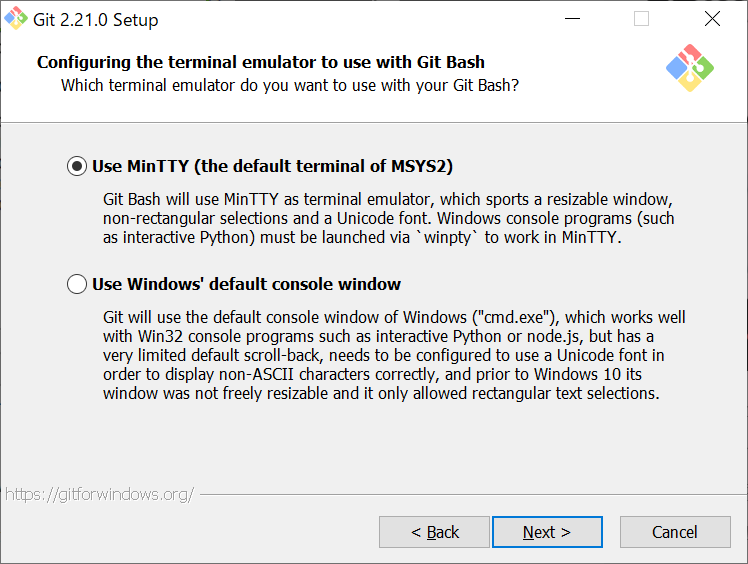
HTTPS接続に何を使うか。デフォルトでNext



8. 改行の取り扱い方式の選択

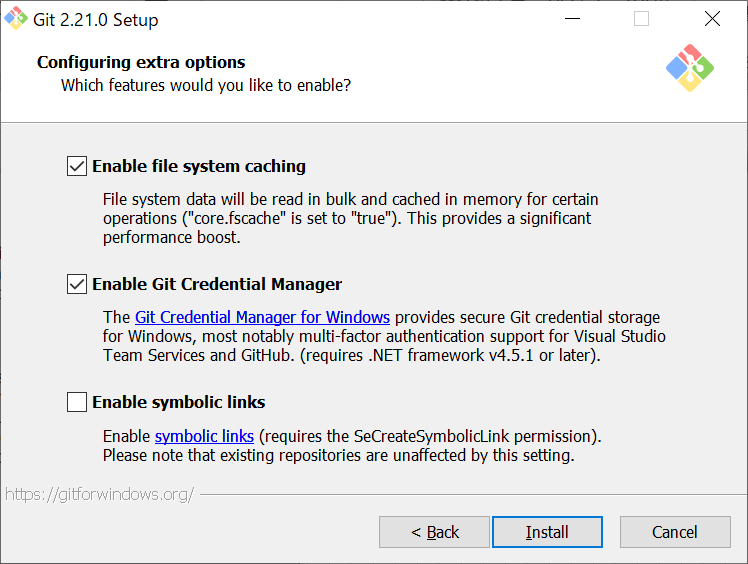


9. Gitで使用するコンソールの選択

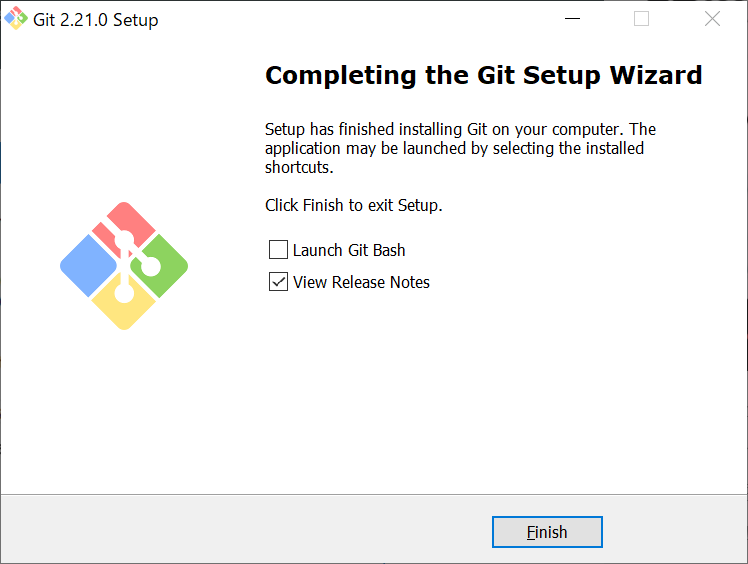


10. 拡張オプションの選択

上のEnable file system cachingは応答速度を改善するfscacheキャッシュを有効にするかしないか。



11. インストール終了



1-3. Gitの設定

1. コマンドプロンプト（Maxはターミナル）を開く

2. 下記コマンドを実行する

git config --global user.name "任意の文字"

git config --global user.email "自分が使うEメールアドレス"

3. 下記コマンドで、エディタの設定を行う

core.editerの後の文字は、使用するエディタによって異なる。

サクラ：git config --global core.editer "'C:/Program Files (x86)/sakura/sakura.exe' -CODE=4"

アトム：git config --global core.editer "atom --wait"

git config --global core.editor 'vim -c "set fenc=utf-8"'

4. diffコマンド、statusコマンド、branchコマンド実行時の色分けを設定

git config --global color.diff auto

git config --global color.status auto

git config --global color.branch auto

5. 日本語の文字化け防止と、ページング設定

git config --global core.pager "LESSCHARSET=utf-8 less"

git config --global core.quotepath false

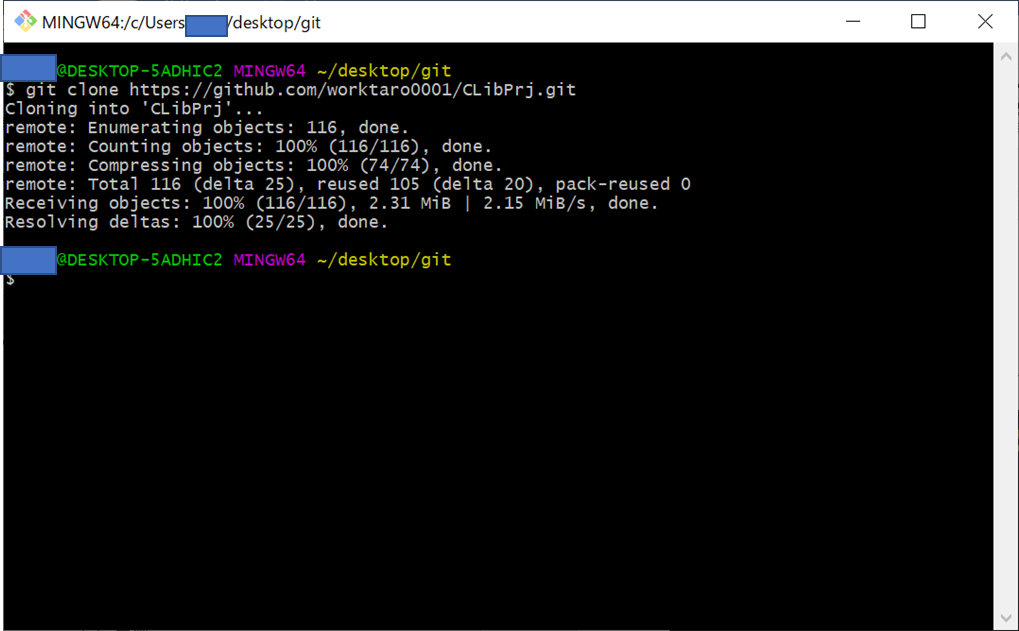
7. 設定した内容を確認

git config --global --list

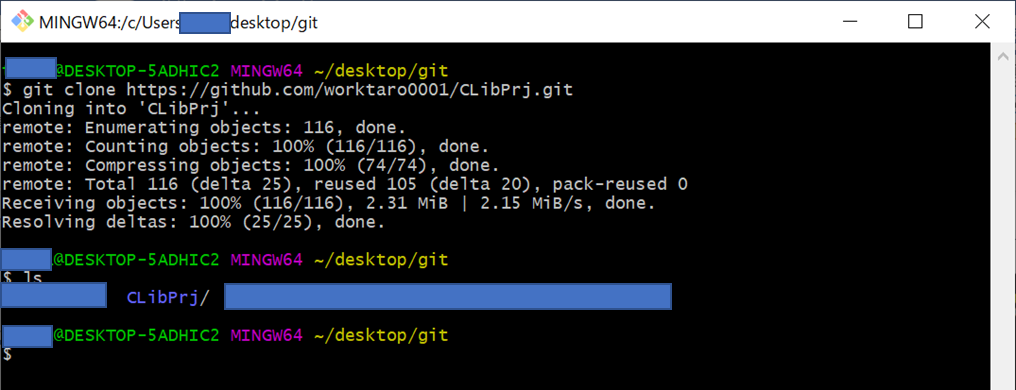
1-4. ローカルリポジトリ環境の構築

1. GitBashを開く

2. git clone [URL] で、ローカルリポジトリを生成



CLibPrjフォルダが作成されていることを確認する。

 ls

3. testブランチを作成する

ローカルリポジトリフォルダ内に移動する。

cd clibprj

testブランチを作成して、移動する。

git checkout -b test

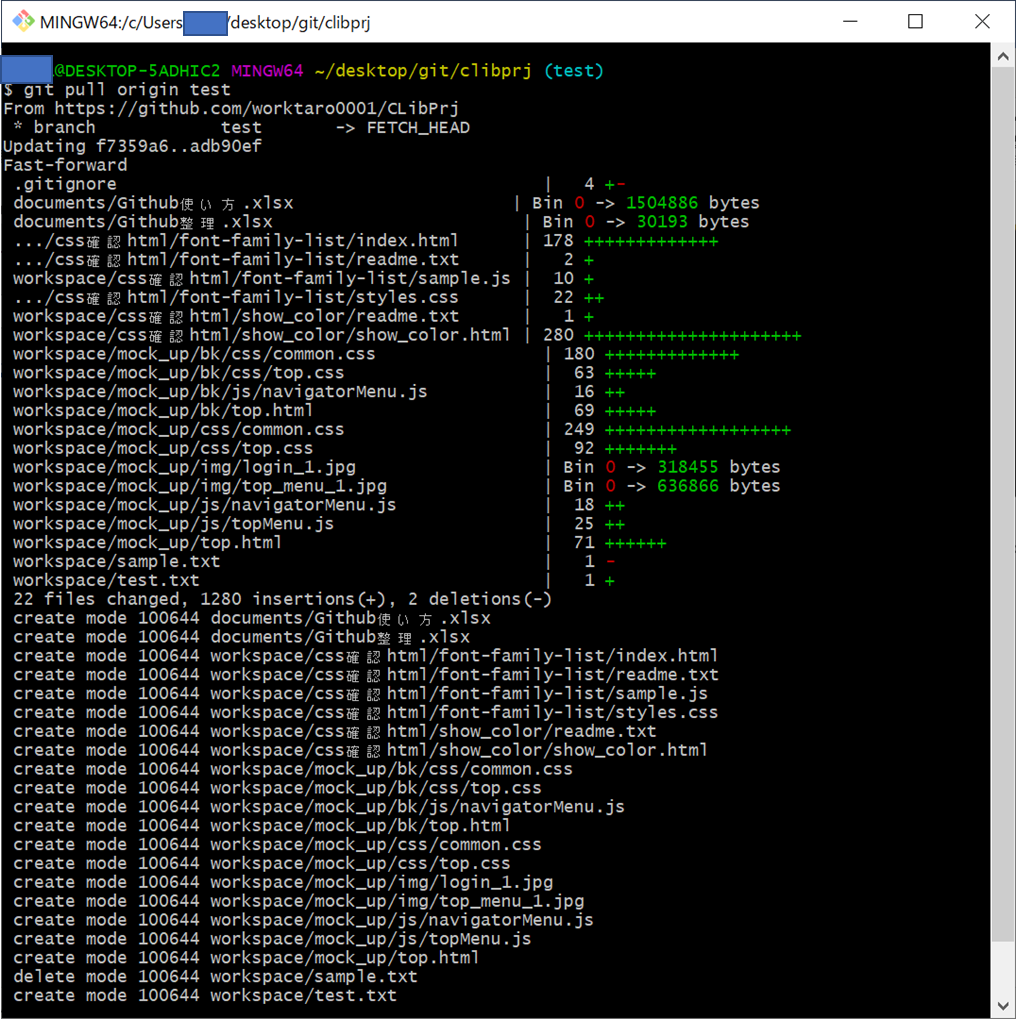
ブランチを確認する

git branch

4. リモートのtestブランチを、ローカルのtestブランチに持ってくる

git pull origin test

※ 実践編の2-3では、fetch+mergeを推奨している。

 　 コンフリクトする可能性もないため、pullを実行している。

2. 実践編

2-1. 目標

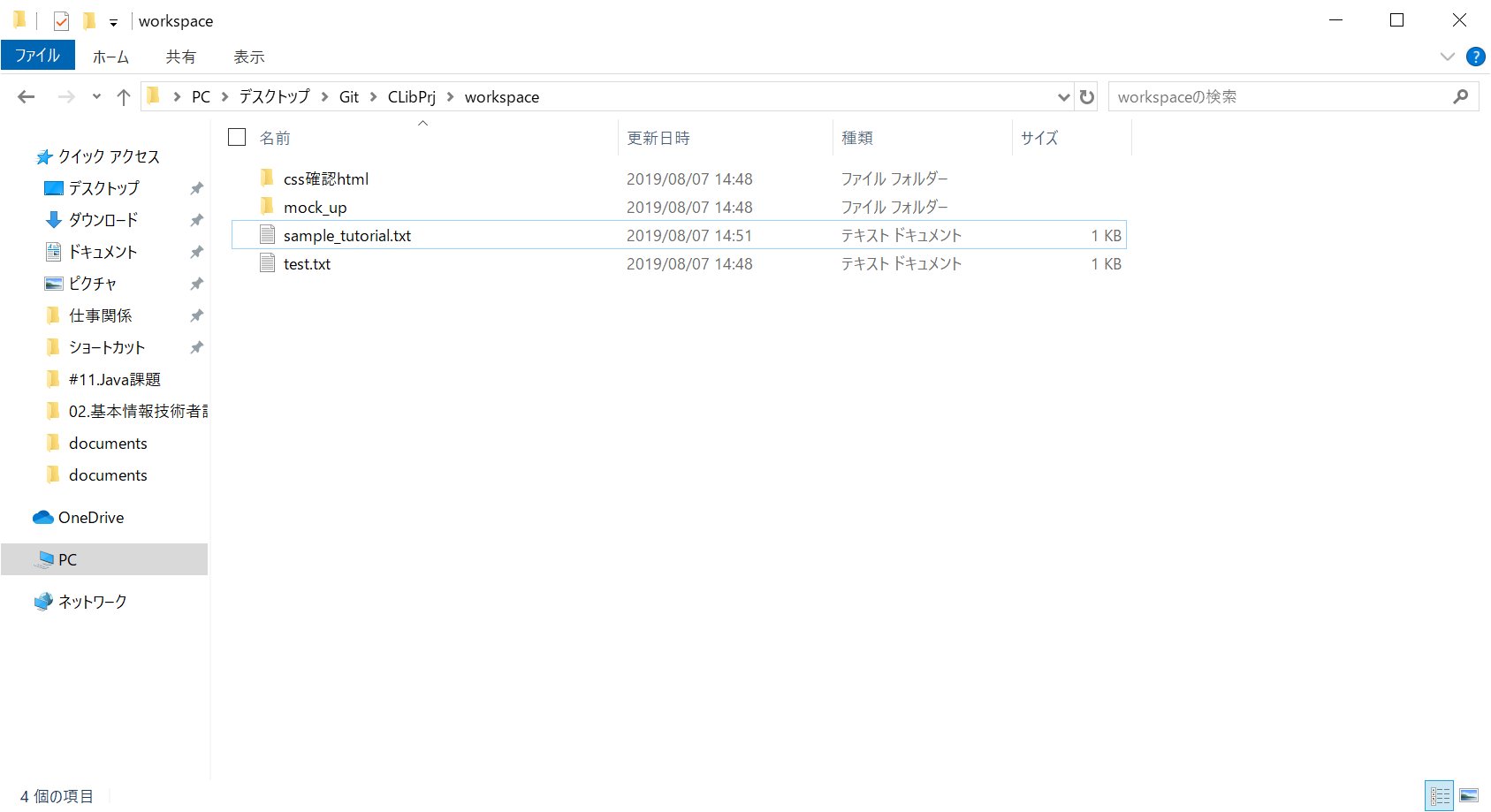
① 新規作成・修正したファイルを、リモートリポジトリに反映させることができる。

② ログの見方がわかる。

③ ソースのマージ方法がわかる。

2-2. ローカルの修正リソースを、リモートリポジトリへ反映する

1. 適当にファイルを作成する。

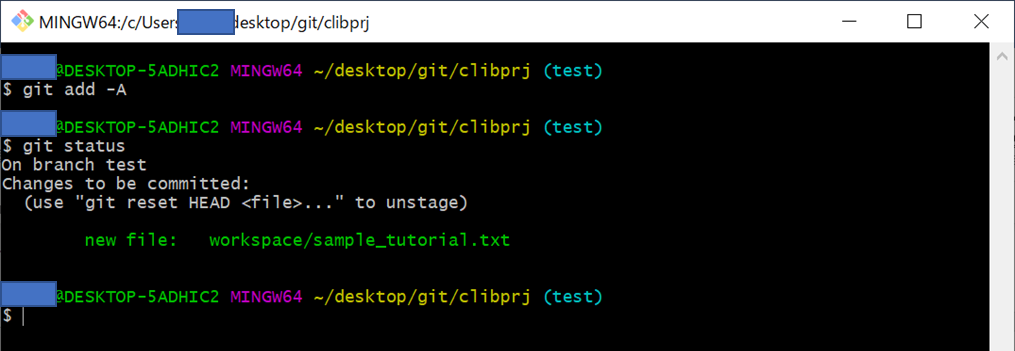


2. インデクスに追加し、状態を確認する。

下記のコマンドを使用する。

git add -A

git status



3. コミットする。

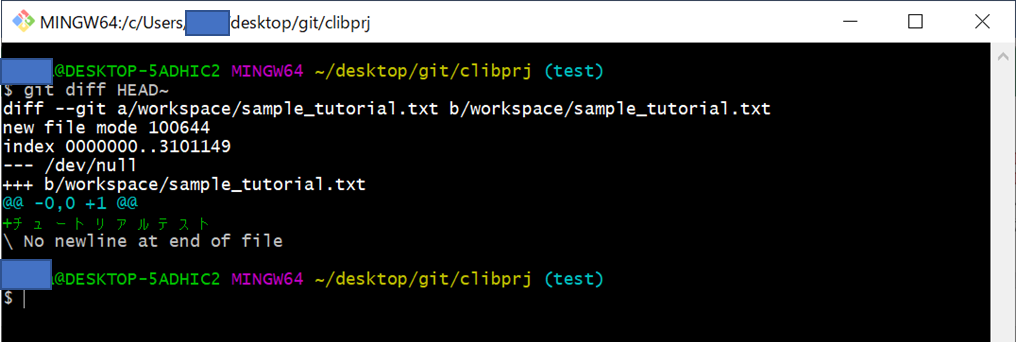
下記のコマンドを使用する。

git commit -m “[メッセージ]”

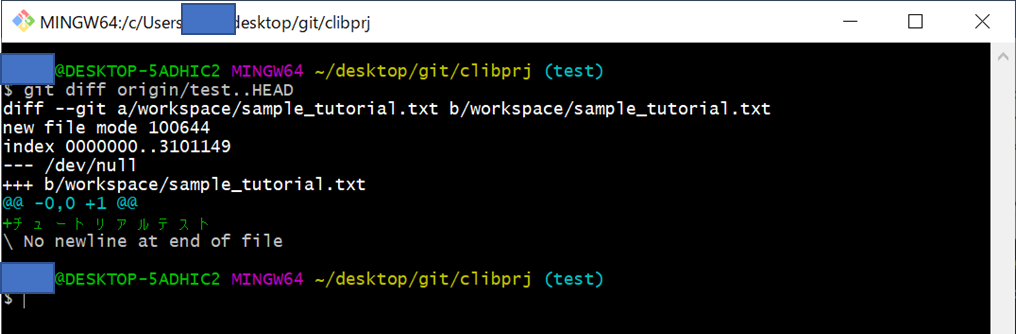
※ git commitのみ実行し、メッセージ記述処理は別途記載でも可

コミットメッセージを記録する。（-m で同時にメッセージを記録できる。）

今回コミット分の差分を確認する。

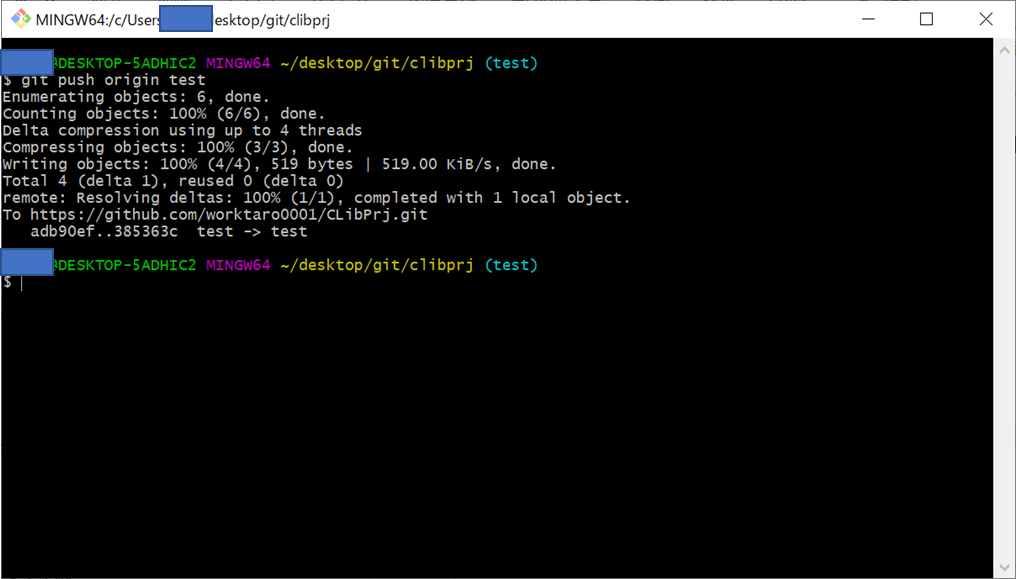
 git diff HEAD~

リモートとの差分を確認する。

 git diff origin/test..HEAD

4. リモートリポジトリへ反映させる。

git push origin test

 ※ -f を使う際は、必ず使用前にメンバーに連絡すること。

2-3. リモートリポジトリの変更点を、ローカルに反映させる

リモートリポジトリの変更点を反映させるには、主に2通りに方法があります。

① git pullコマンドを使用する方法

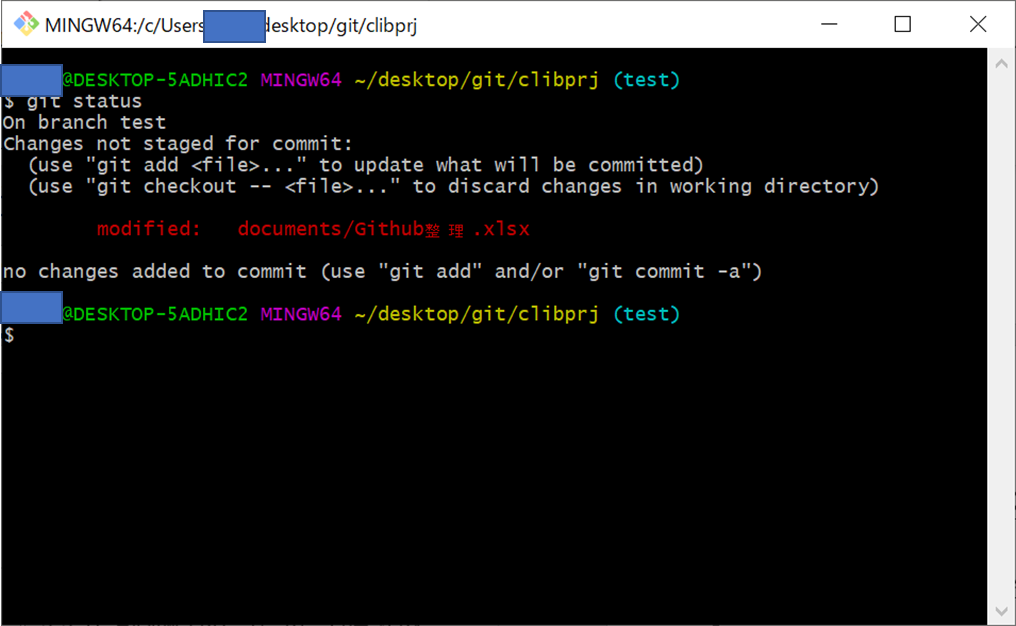
② git fetch + git mergeコマンドの２つを使用する方法

本書では、**fetch + merge**を併用したものを使用します。pullは、fetch + mergeを**一括**で行うコマンドです。便利ですが、コンフリクトの対応をする際に、fetchは必ず覚えることになります。

なので、１つ１つのコマンドを理解する意味でも②の方法を採用します。

異論は、認めます。

1. 現在のワーキングツリーの状態を確認する

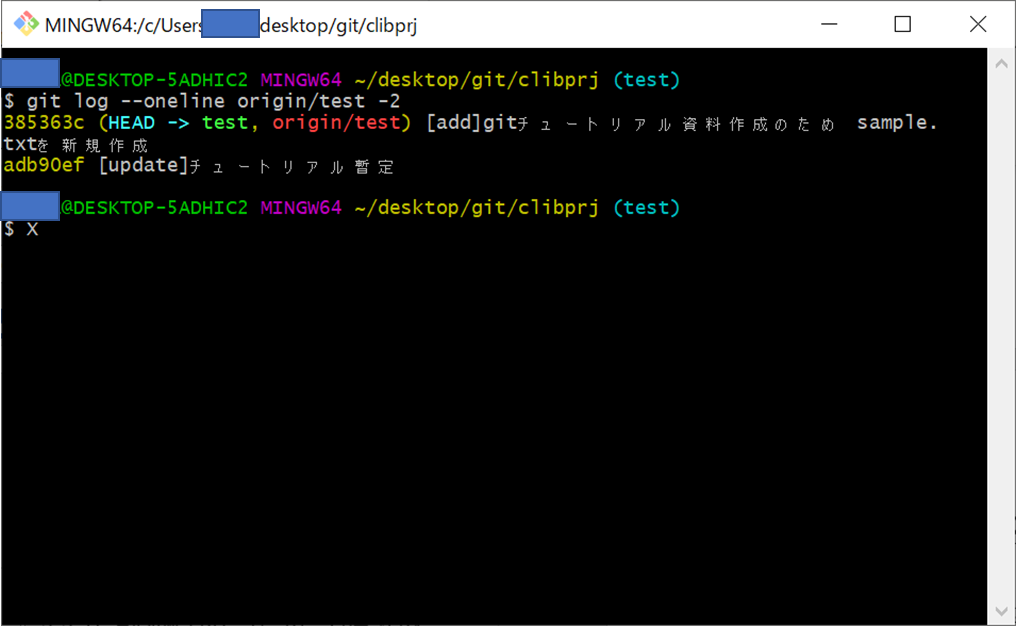
 git status

※ここでは差分がでているが、本来はでてはいけない。

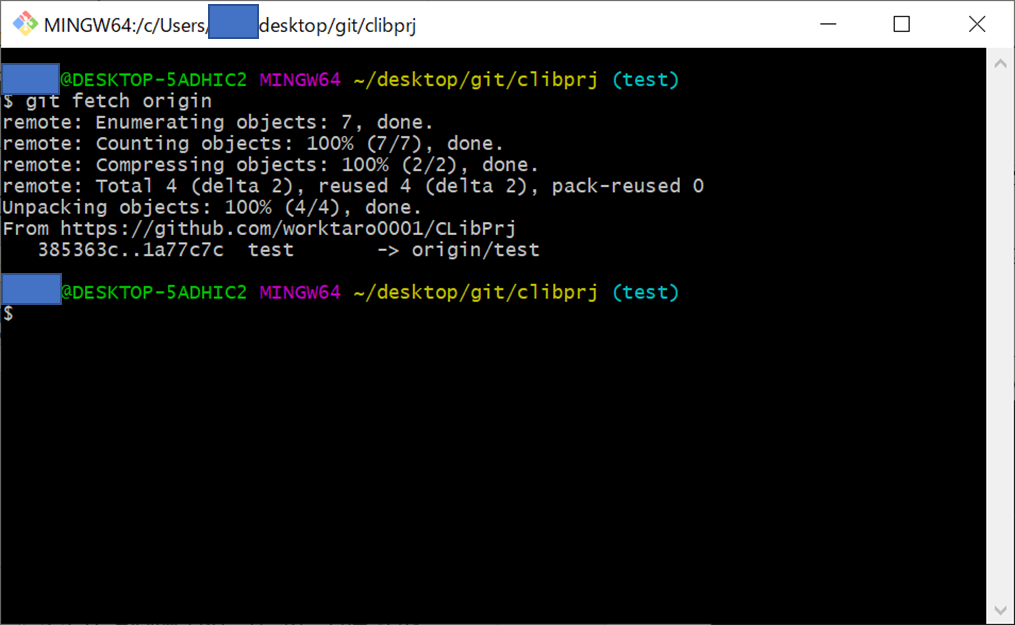
もし差分がでた場合は、ステージング環境にあげて(add)、コミットまで行う。

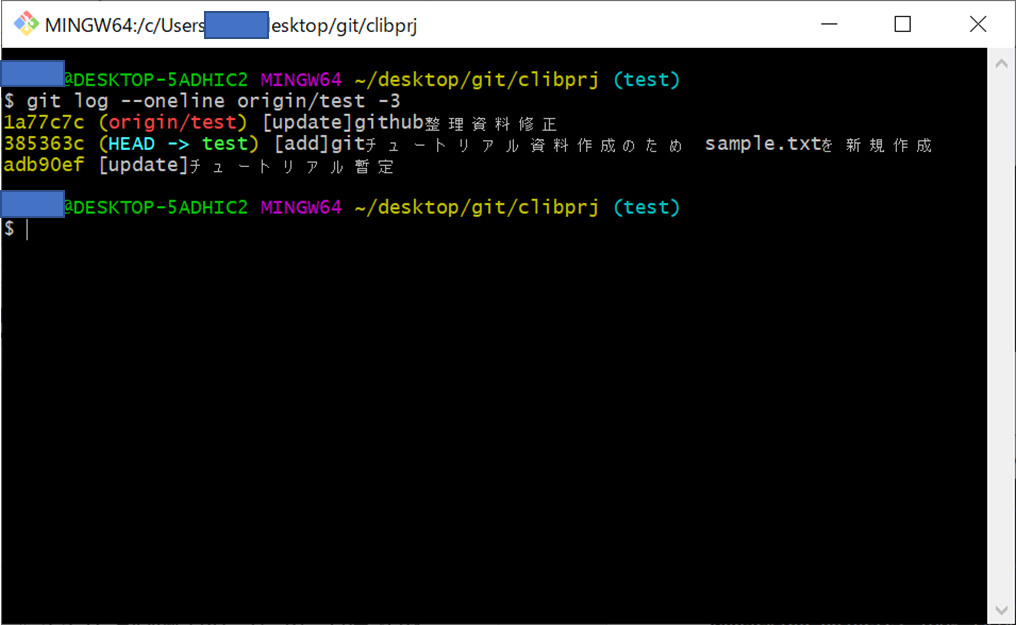
2. リモートリポジトリとの差分を確認する

更新前のログを確認する。

 git log origin/test （他のオプションは、お好みで）

リモートの状態のみを更新する。

 git fetch origin

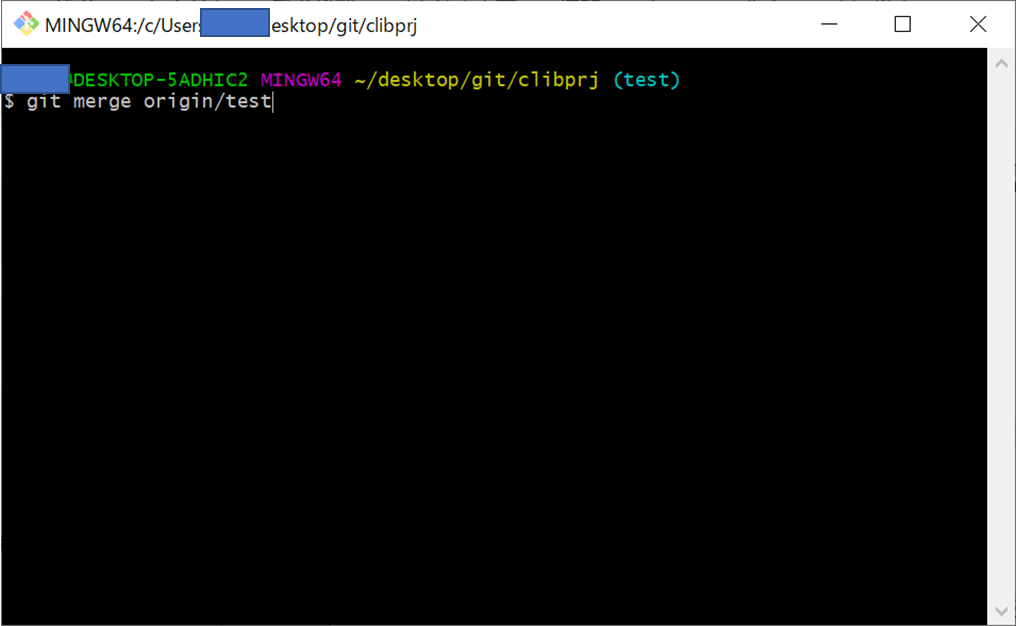
更新後のログを見る。

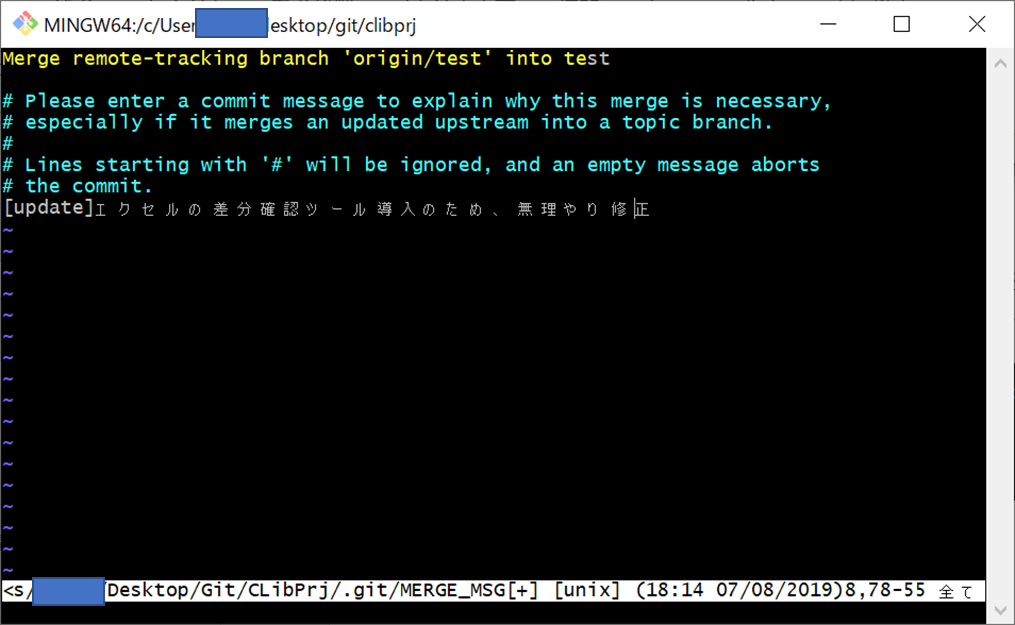
ローカルの状態は

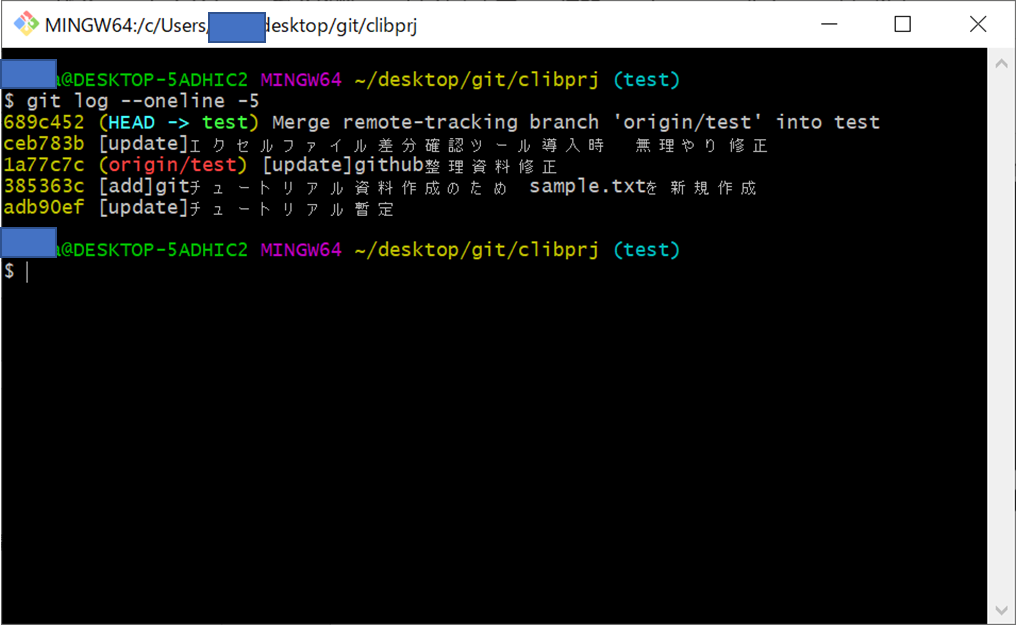
更新されていない

3. リモートリポジトリの変更ファイルを、ローカルに反映させる

2のfetch後のlogコマンドで、リモート上の変更点が、ローカル上の変更点に影響がないことを確認しておくこと。

 git merge origin/test

マージログを記載する。

ローカルリポジトリのログを確認する。

3. トラブル編

3-1. 目標

① 修正したファイルを、修正前の元の状態に戻すことができる。

② ステージング環境にあげたファイルを取り消すことができる。

③ コミットしたファイルを取り消すことができる。

④ pushしたファイルを取り消すことができる。

⑤ pullしたファイルを取り消すことができる。

3-1.5. トラブル時に必要な基礎知識

トラブルの対応編に入る前に、事前に覚えておいた方がいい知識を整理してみました。

自分なりにまとめただけなので、何か異論があればガシガシご意見ください。

● ローカルリポジトリで作業している時のイメージ

まずは、ローカルリポジトリでファイルを操作している時のイメージを掴みましょう。

通常、ファイルを修正しただけで反映されるのは、**ワーキングツリー**と呼ばれる環境のみです。

このイメージ図では、ファイルAが修正されてファイルA+となり、さらに修正されてファイルA++となっています。しかし、まだaddもcommitもしていないので、**ステージング環境**と**HEADの位置**は変わっていません。

ということで、これから下線を引いた用語を念頭において、トラブルに対応する方法を見ていきましょう。



3-2. 修正したファイルを、修正前の状態に戻す

前回の図の続きですが、図の中では、[ファイルA]を修正してできた[ファイルA+]を、さらに修正して[ファイルA++]としました。

ここで、この[ファイルA++]を[ファイルA+]に戻したい状況になったとします。

その場合、以下のコマンドを使用します。

git checkout HEAD .

（※今回は、現ディレクトリ配下全てが対象。ファイルを限定したい場合はパスを指定）

もしくは

git reset --hard

上記のコマンドを実行することで、ローカルリポジトリの状態は、図中の下部にあるHEADの位置に、ワーキングツリーまるごと戻ります。

3-3. ステージング環境にあげたファイルを取り消す

 次は、ステージング環境にあげたファイルを取り消す方法です。まず、ステージング環境にファイルがあげられた際のイメージとしては、以下のようになります。

今回は、ステージング環境のindexの状態を、HEAD時点の状態に戻したいとします。

その場合、以下のコマンドを使用します。

git reset HEAD .（※前回同様、現ディレクトリ配下全て）



もし、修正ファイルも戻したい場合は、オプション--hardを指定しましょう。

3-4. コミットしたファイルを取り消す

次に、コミットしたファイルを取り消す方法です。

commitを実行した場合の状況図は、以下のようになります。



気づいているかもしれませんが、commitをすることでHEADの位置が変わっています。

こうなると、先程のように

git reset –hard HEAD

とするわけにはいきません。HEADはさっきcommitした状態なので、何も戻りません。

その場合は、以下のように使用します。

git reset –soft HEAD~

（※--hardでもいいですが、ここではcommitの状態だけを戻したいので、softに）

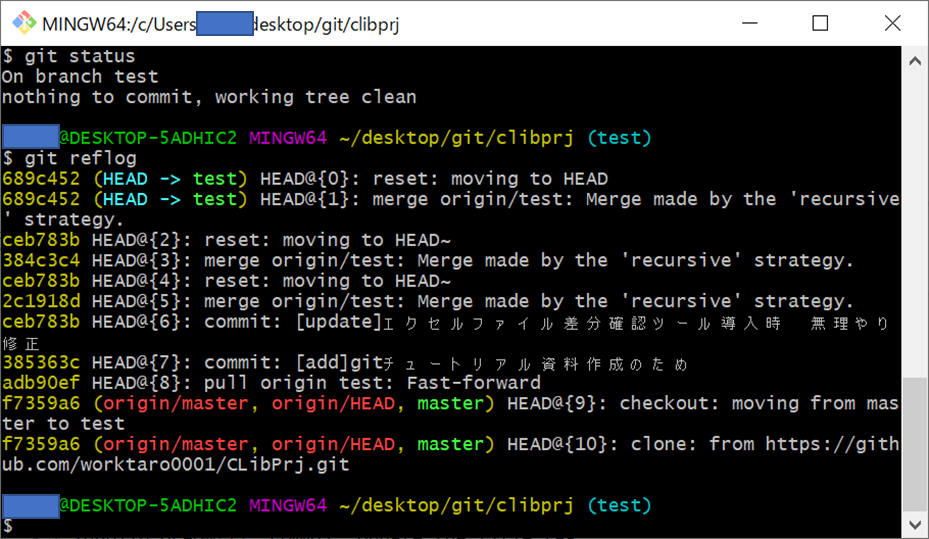


もし、--hardを指定すると、以下のようになります。



ワーキングツリーからステージング環境、HEADの位置まで、全て元通りになります。強いです。

ちなみに、git reflogというコマンドを使用すると、HEADのの遷移を全て見ることができます。



この**HEAD@{7}**とか**HEAD@{4}**とかを指定しても、元通りに戻ります。便利です。

3-5. pushしたファイルを取り消す

やっちまったなぁパターンです。以下のサイトがわかりやすいです。

<http://www-creators.com/archives/2020#_push>

主に２つの方法があります。

① 過ちを隠す方法

② 過ちを認める方法

① 過ちを隠す方法

以下の流れで行います。

・１回前のコミット状態に戻す。

git reset –hard HEAD~

・戻した状態のコミットを、強制pushする。

git push -f origin [ブランチ名]

この、-fがキモです。全てを上書きします。

この-fを指定しなければ、古いコミット状態をpushしようとしているため、コンフリクトではじかれますが、この-fを指定すると、強制的に指定したコミット状態に上書きします。



先程のURL先のサイトでも紹介されていますが、この方法はコミットログも全て前の状態に戻しちゃうので、できれば②の方法をとってほしいです。。。

② 過ちを認める方法

以下の流れで行います。

・最新のコミットを打ち消し、前回のコミット状態を新たにコミットする。

git revert HEAD

・戻した状態のコミットを、強制pushする。

git push origin [ブランチ名]

この場合は、前回のコミット状態を、新たにコミットして作成しています。

過去に戻らないため、-fは使用せずにそのままpushできます。ログが汚くなるのが難点です。



3-6. pullしたファイルを取り消す

これに関しては、以前に説明したコマンドで戻すことができます。

git reset --hard HEAD~

しかし、これはpullがコンフリクト等で中断されなかった場合です。もし、pull中にエラーが発生して、それがコンフリクトによるものだった場合は、以下の２つのうちいずれかを実施します。

① コンフリクトを解決して、マージを続行する

② マージを中断する

①については、その時対応していきましょう。

②については、git merge --abort

を実行することで、中断できます。

4. 補足 – コマンドオプション集

4-1. git diff

4-1. git log

4-1. git checkout

4-1. git add

4-1. git commit

4-1. git pull

4-1. git push

4-1. git reset

4-1. git fetch

4-1. git merge

4-1. git rebase

4-1. git ls-remote

5. 補足 – エクセルの差分確認ツール

5-1. 目標

① エクセルファイルにおける差分が確認できるようになる。

5-2. ツールの導入

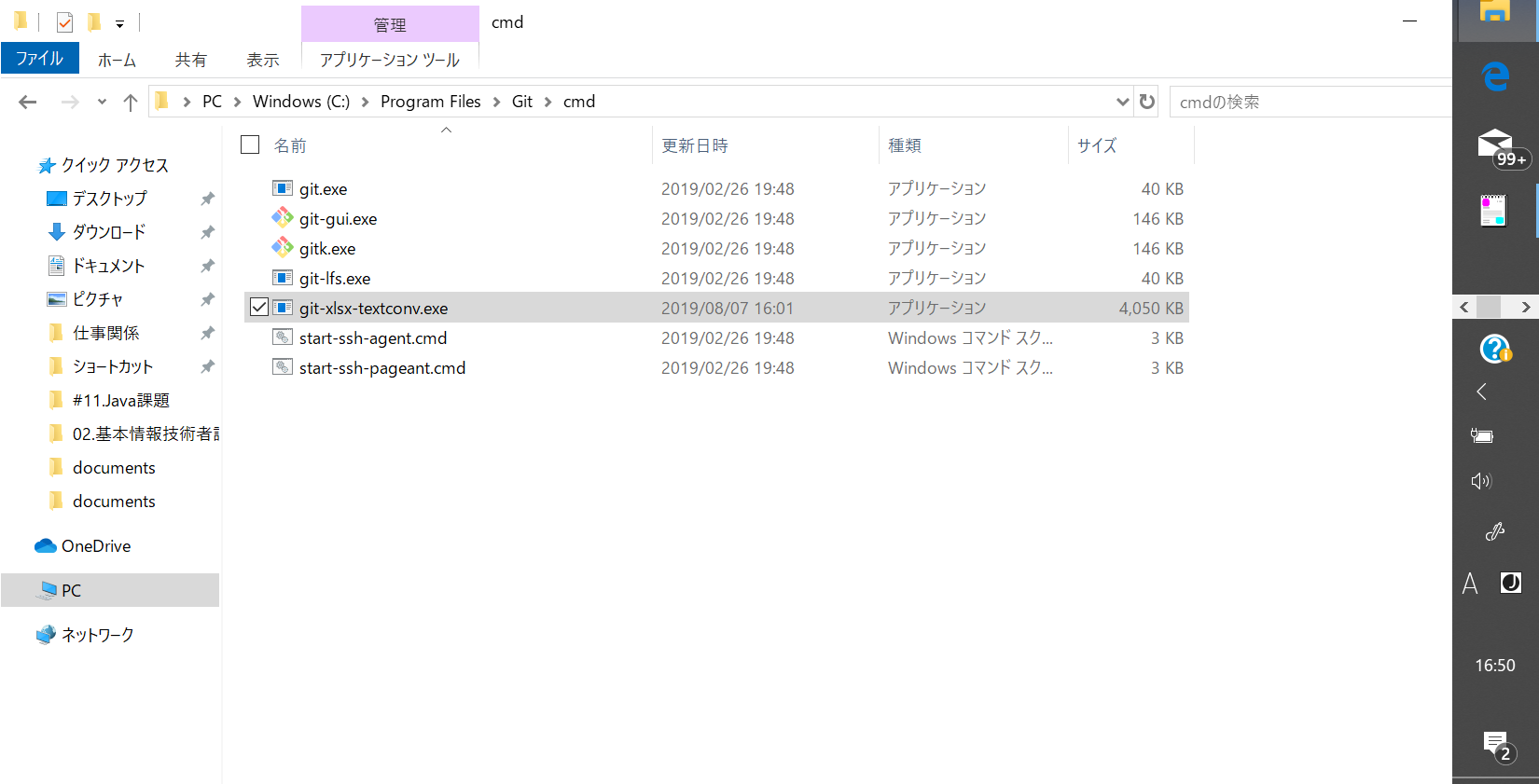
1. 下記URLにアクセスする

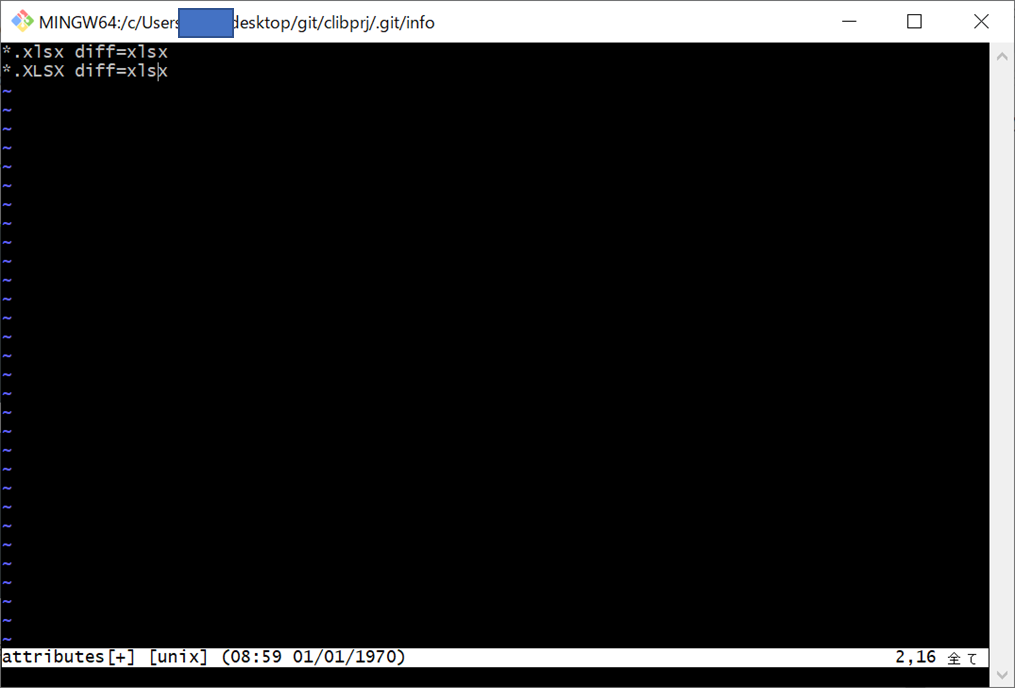
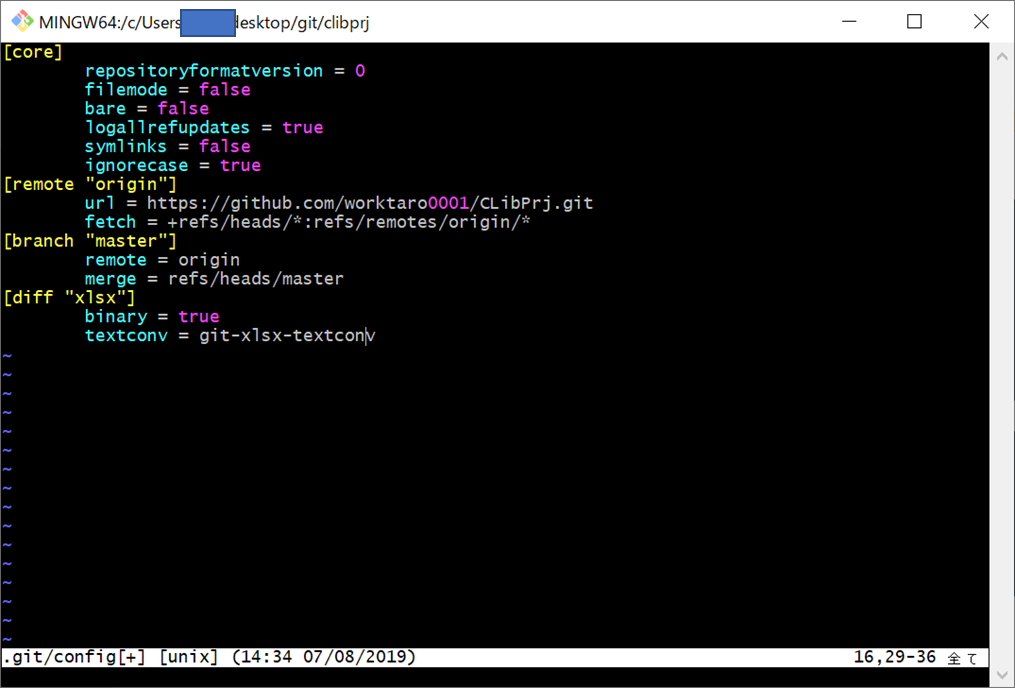
<https://github.com/shibukk/git-xlsx-textconv/releases/tag/v1.0.0>

2. git-xlsx-textconv\_windows\_amd64.zip をダウンロードする

3. ダウンロードしたzipファイルを展開して、gitインストール時のフォルダに追加する

おそらく、C:\Program Files\Git\cmd フォルダがデフォルト（のはず）。

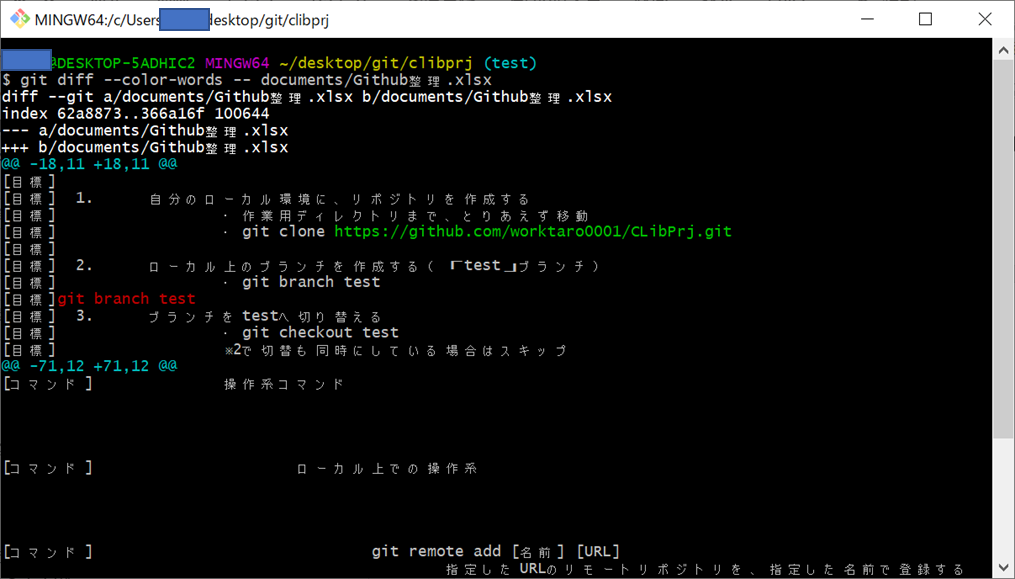


4. .git/configファイルと、.git/info/attributesファイルを編集する

5. git diffで差分を確認する

左端の[～～]がシート名

差分のある文字は、色がついて表示される。

※ウイルススキャンが走ると正常に動作しないため、ここだけ外すとか個別設定をする