G^2 のための Git 概要

制作者: 原凪利

制作日: 2024/6/8

目次

1 はじめに 1
2 Git とは 1
2.1 バージョン管理システムとは 1
2.2 分散型と集中型の違い 1
3 構造・基本的な使い方 1
3.1 共同開発における全体の構造 1
3.2 プロジェクトへの参加(clone) 3
3.3 編集の仕方(add、commit、push、pull (fetch、merge)) 3
4 ブランチ 5
4.1 ブランチとは 5
4.2 競合(コンフリクト) 6
5 用語の整理 6
6 リモートリポジトリの作成・プロジェクトの初期化 (実践) 7
6.1 リモートリポジトリの用意 7
6.2 プロジェクトの初期化 8
7 プロジェクトへの参加(clone の仕方) (実践) 9
8 編集方法(リモートリポジトリへの反映) (実践) 11

1　はじめに

今回は細かいことを省略して概説する。

しかし、Git を含むバージョン管理システムは社会に広く普及しているため、各自で理解を深めることを推奨する。

また、本サークルにおいて Git、及び GitHub の、少なくとも今回の内容は必修に値するものとする。

内容として、2 章から 5 章は知識編、6 章以降が実践編となっている。

なお、この解説は Git、GitHub、GitHub Desktop、Unity を用いた開発を想定している。

2　Git とは

Git とは、分散型のバージョン管理システムである。

2.1　バージョン管理システムとは

バージョン管理システムとは端的に言えば、ファイルへの変更履歴を記録しておくシステムのことである。

2.2　　分散型と集中型の違い

バージョン管理システムは分散型と集中型に大別される。

集中型は、プロジェクトメンバーで共有しているオンライン上のファイル群にアクセスして、直接変更を加える。

分散型は、プロジェクトメンバーで共有しているオンライン上のファイル群をそれぞれの開発者のコンピュータ内にコピーして、コピーしたファイル群に変更を加える。そして、自身のコンピュータ内で行った変更が正常に動くことを確認してから、それらをオンライン上にアップロードする。

3　構造・基本的な使い方

3.1　共同開発における全体の構造

バージョン管理システムは、オンライン上にあるファイル群の保管庫であるリモートリポジトリに変更点のみをアップロードすることで、共同開発を効率的に行うことを実現している。

共同開発を行う際の構造を全体的に広く見たときの概要を以下の図 1 に示す。

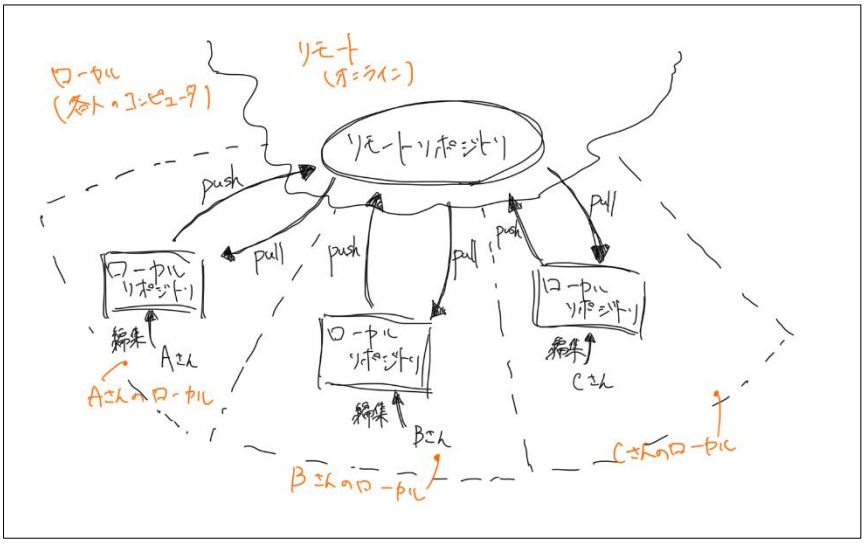

図 1: 共同開発における構造概要

Git では、オンライン環境のことをリモート(remote)、各人のコンピュータ内の環境のことをローカル(local)と呼んでいる。また、ファイル群、ソースコード群を管理する保管庫のことをリポジトリ(repository)と呼んでいる。

Git における共同開発では、リモートリポジトリからコピーしてきたローカルリポジトリを各開発者が編集する。ここで、自分のコンピュータ内にローカルリポジトリが無い状態でリモートリポジトリから最初にコピーすることを、クローン(clone)と呼んでいる。clone の概要は同章 2 項で説明する。

各開発者は clone してきたローカルリポジトリ内で内容の変更を行い、問題がなければリモートリポジトリにアップロードする。このアップロードのことを、プッシュ(push)という。

また、他者が行った変更を自身のローカルリポジトリに反映させることを、プル(pull)という。ただし pull に関して、厳密に言えば少し違う。その詳細、及び push、pull の概要は同章 3 項で説明する。

3.2　プロジェクトへの参加(clone)

この項では、プロジェクト(リモートリポジトリ)が既に存在していて、それに参加することを想定する。リモートリポジトリの作成方法に関しては、実践編の方で説明する。

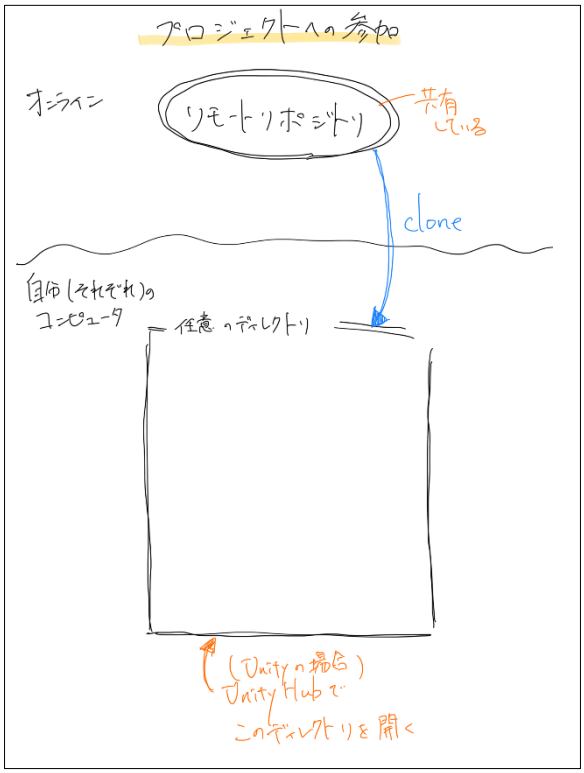

図 2: プロジェクトへの参加概要

プロジェクトへの参加方法の概要を、図 2 に示した。

まずはローカルリポジトリを準備する必要がある。

そのため、前項で述べた clone を行う。また、Unity で開発を行う場合は、Unity Hub を用いて clone してきたプロジェクトを開く必要がある。

3.3　編集の仕方(add、commit、push、pull (fetch、merge))

この項では、前項で clone してきたローカルリポジトリを編集し、その変更をリモートリポジトリに反映させる方法、また、他人が変更したリモートリポジトリを自身のローカル環

境に反映させる方法を紹介する。

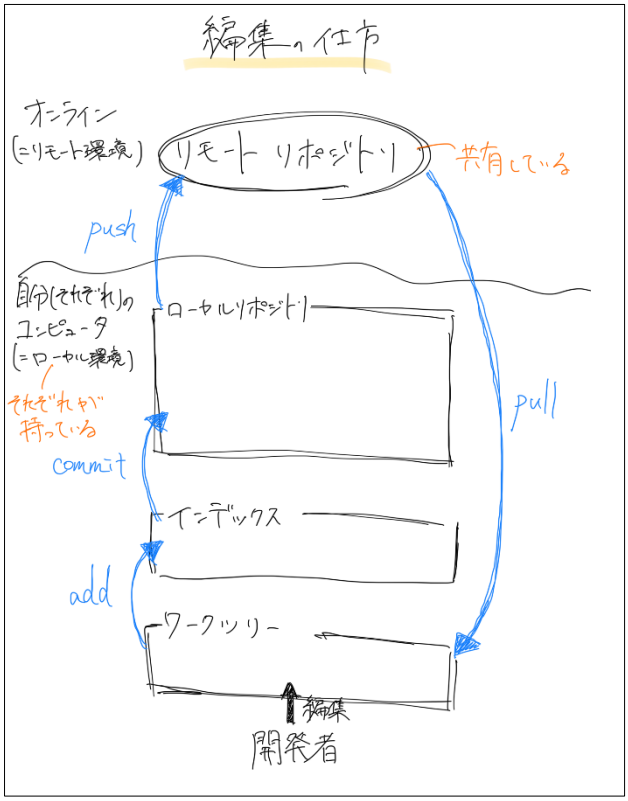

図 3: 編集の仕方概要

編集の仕方概要を図 3 に示した。

リモートリポジトリを clone すると、ワークツリー、インデックス、ローカルリポジトリという領域がローカル内に自動的に作られる。

ワークツリーは自身の編集する対象となるディレクトリのことであり、インデックスは行った編集の内どのファイルの変更を反映するかを控えておく場所のことである。

また、ローカルリポジトリはローカル内におけるリポジトリのことであり、ワークツリーとは別のものである。ワークツリーに対して行った変更は直接的に保存されることはないが、ローカルリポジトリに対して行った変更はコメントを付けて保存することになる。

開発者はワークツリーで編集を行い、行った変更の内反映させたいものをインデックスに add する。ひとつの変更が完了しそれらの変更をインデックスに add したら、変更の内容を記してからローカルリポジトリにそれらをコミット(commit)する。

ローカルリポジトリに異常がないことを確認したら、リモートリポジトリにそれらを push する。

また、他人がリモートリポジトリを変更しているか確認するためにフェッチ(fetch)し、更新があったらそれをワークツリーにマージ(merge)する。ただし基本的には、fetch と merge

をあわせたショートカットである pull が使われる。

全くもって穿っていない極めてひねくれたまとめ方をすると、ワークツリーで変更をし、そのうち反映したいものをインデックスに add し、そのひと纏まりの変更を名前付きでローカルリポジトリに commit する。このとき、ローカルリポジトリには変更履歴が保管されている。そしてその変更履歴を全体で共有するために、リモートリポジトリに push する。するとリモートリポジトリに変更履歴が反映され、その変更が実際に行われる、といった感じである。

4 ブランチ

4.1 ブランチとは

ブランチとは、バージョンを分岐させることが出来る機能のことである。

機能の追加、バグ修正などを独立、並行して行えて、例えばもし機能 A ブランチの制作中により優先度の高い他の機能 B を追加する必要がある場合、機能 A のない状態のマスターブランチ(メインとなるブランチ)から機能 B のブランチを作成することが出来る。

また機能 A が不要になった場合でも、マスターブランチに結合していなければ影響が出ることがない。

ここで、図 4 にブランチのイメージを示した。

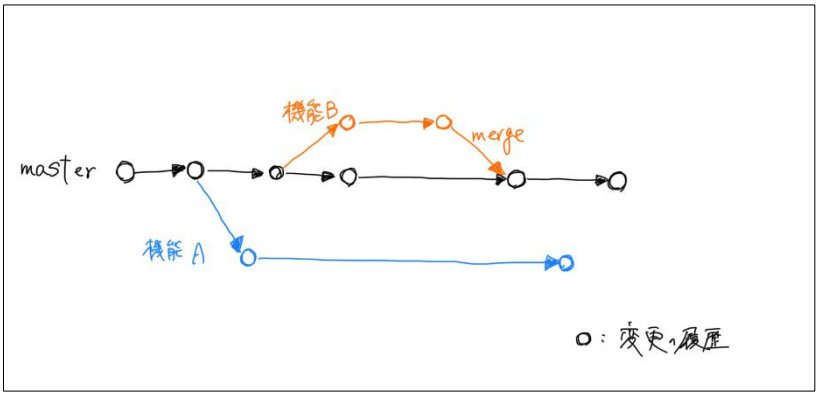

図 4: ブランチのイメージ

ブランチを他のブランチに結合するには、merge を行う必要があり、これには要求(pull request)が必要である。この大きな目的は、競合の発見である。競合に関しては、次項で解説する。

また別の理由として、merge を許可する側のブランチの管理者が、merge しても問題ないか人的に確認をするためということが考えられる。例えば既存のプロジェクト A に新入社

員 C が参加し、マスターブランチを管理しているベテラン B に研修としてミニゲームを作るよう依頼された場合を想定する。C はミニゲーム用のブランチを作成し、その中で目的のミニゲームを完成させ、pull request をマスターブランチに対して送信した。このとき、ミニゲームに潜在的で致命的なバグがあった場合、確認があることで B さんはそれを発見することが出来る。

また、GitHub では pull request に対してコメントを残すことが出来るため、修正して欲しい部分をコメントすることが出来る。

さらに用語として、別のブランチに移動することをチェックアウト(check out)という。

4.2 競合(コンフリクト)

ブランチの merge においては、競合ということが起こることが往々にしてある。

競合は、複数のブランチで同じ部分を変更し、それらを merge しようとした際に発生する。

競合が発生したら、merge を許可する側のブランチがどの変更を採用するのか決定する必要がある。

5 用語の整理

Remote(リモート)	：オンラインのこと
Local(ローカル)	：開発者のコンピュータのことであり、開発者の数だけある
Repository(リポジトリ)	：ファイルを保管する保管庫
Work Tree(ワークツリー)	：開発者が編集する直接の対象
Index(インデックス)	：反映する変更の置き場所であり、ここにあるファイルは staging(ステージング)されていると言う
Clone(クローン)	：リモートリポジトリからローカルリポジトリを作成すること
Add	：反映する変更を index に追加すること
Commit(コミット)	：コメントを付けて index にある変更を保存すること
Push(プッシュ)	：ローカルリポジトリからリモートリポジトリに変更をアップロードすること
Pull(プル)	：ワークツリーをリモートリポジトリの最新の状態に更新すること(fetch + Merge)
Fetch(フェッチ)	：リモートリポジトリの最新の状態を読み込むこと
Merge(マージ)	：Push の一部としては、fetch した最新の状態をワークツリーに反映させること ：ブランチにおいては、他のブランチと結合すること

Branch(ブランチ)　　：バージョンを分岐させる機能

6　リモートリポジトリの作成・プロジェクトの初期化（実践）

これ以降の章では、実際の手順を紹介していく。

6.1　リモートリポジトリの用意

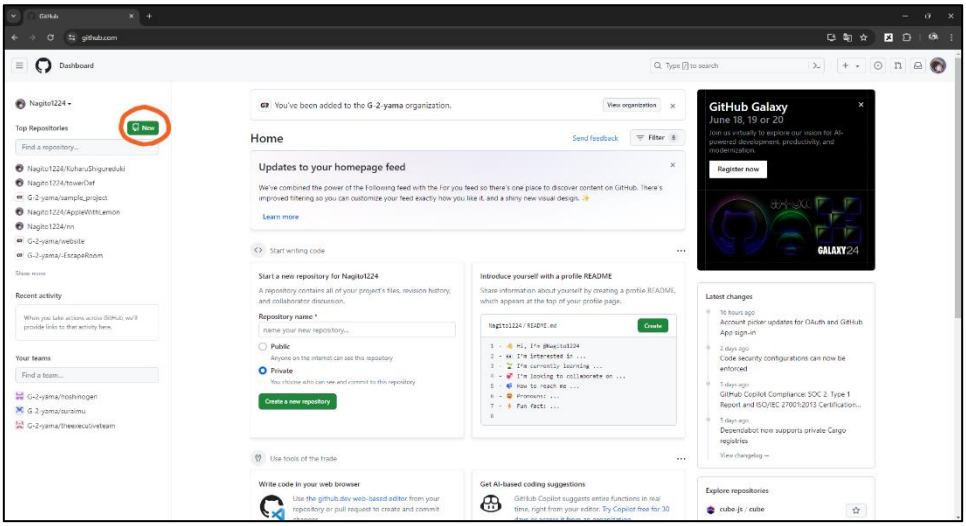

図 5: GitHub　ホーム画面

まずはプロジェクトを始めるために、リモートリポジトリの作成と初期化を行う。

リモートリポジトリを作成するには、まず GitHub のウェブサイトにアクセスし、ログインをする必要がある。また、アカウントを持っていない場合は作成をする。

ログインが完了すると、図 5 のような画面になる。

ここで新たなリモートリポジトリを作成するには､橙色の丸で示した左上の New を押す。

すると、図 6 のような画面になる。

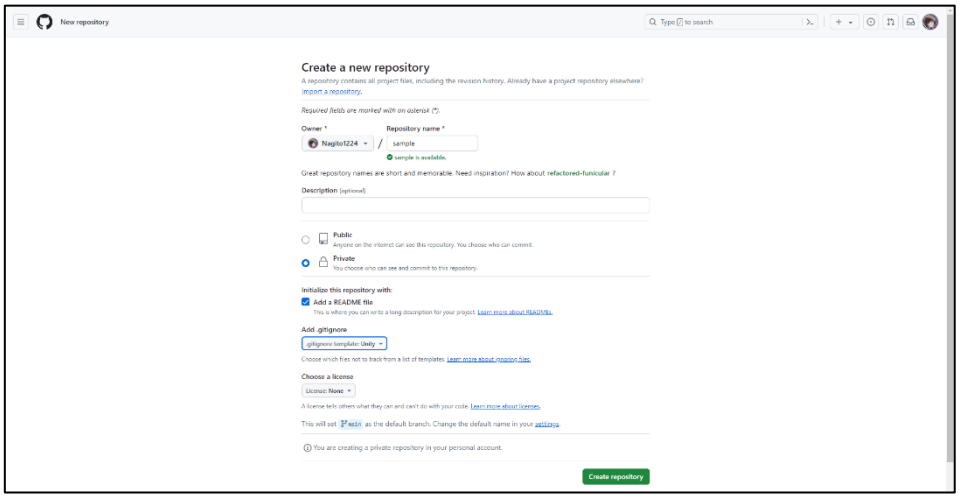

図 6: GitHub リモートリポジトリ作成画面

Owner にはリモートリポジトリの所有者、Repository name にはリモートリポジトリの名前、Description にはリポジトリの説明(任意)を入力する。

また、Public はリポジトリの中身を全ての人が閲覧可能な設定で、Private は許可した人のみ閲覧可能という設定である。

Add a README file は説明書のようなものを含めるかどうか、Add .gitignore は除外したいファイル群のテンプレートの選択、Choose a license は他人がリポジトリのファイルを利用する際のライセンスの選択をする。gitignore ファイルは、どのツールに最適化するか選択する場所だと考えてもらって構わない。例えば Unity を用いて制作を行う場合、ここで Unity を選択しておけば、Unity が用意したファイルの内、リポジトリで管理する必要のないファイルを除外してくれる。

入力を終えたら、Create repository を押す。

これで、リモートリポジトリの作成は完了した。

6.2 プロジェクトの初期化

Unity を用いる場合、Unity が用意するファイルをリモートリポジトリに含める必要がある。

まず、リモートリポジトリの clone を行う。クローンの手順については冗長性を減らすために、7 章を参照して行うものとする。

次に、Unity のプロジェクトを作成する。この手順に関しては Unity の基本操作について説明した資料を見て欲しい。

この時点でローカルには Unity のプロジェクトディレクトリと、クローンしたリポジトリ(ワークツリー)が存在する。

ここで、Unity のプロジェクトディレクトリの中身を全てワークツリーの中に移動する。

この段階で GitHub Desktop を見ると、28 個の変更がインデックスの中に自動で追加されていることが分かる(インデックスに入っている変更の数に多少の変動はあるかもしれないけれど、10000 を超えるような場合は流石に何かミスっていると思ったほうが良い)。このときの GitHub Desktop の様子を図 7 に示す。

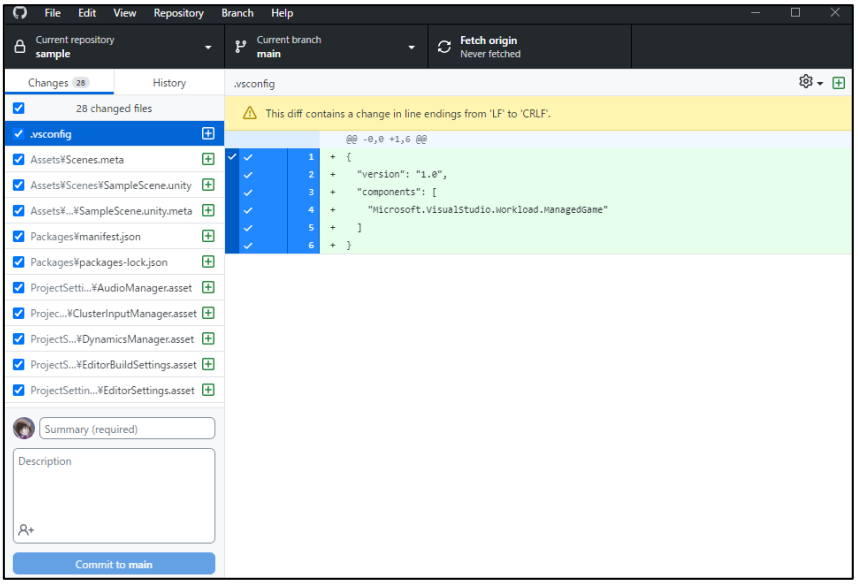

図 7: 最初の commit 時の GitHub Desktop 画面

ここで init commit や first commit など適当なコメントとともに commit を行ってから、GitHub Desktop 中央よりやや右の上の方にある Push origin を押すと、プロジェクトの初期化が完了する。

commit、push の詳細な方法に関しては 8 章を参照することとする。

最後に、Unity Hub のプロジェクトの中から先程作成したプロジェクト(ファイルを移動したためにもう開かないプロジェクト)を削除して、”追加”から先ほど Unity プロジェクトを移行した先のディレクトリ(ワークツリー)を開けば編集を開始できる。

7　プロジェクトへの参加(clone の仕方) (実践)

この章では、リモートリポジトリが用意されていてローカルリポジトリがまだない状態から、ローカルリポジトリを作成する方法について紹介する。

まずは GitHub の web サイトの中で参加したいリモートリポジトリのページに移動する。

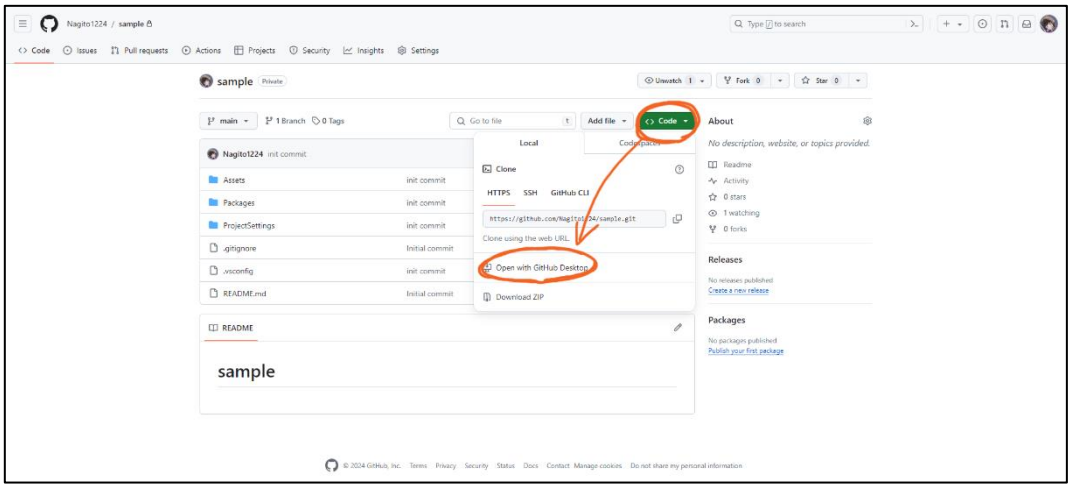

図 8: GitHub のリポジトリ画面(clone)

その中で、図 8 に示したように、”Code”を押してから、”Open with GitHub Desktop”を押すと、図 9 のような画面に自動的に移動する。自動的に移動しない場合は、GitHub Desktop 左上の”Current repository”から、”Add”、”Clone a repository”を押すか、あるいは”Clone a repository from the Internet…”を押して、上の入力欄に web サイトのリポジトリページ、”Code”の HTTPS の下にある URL をペーストすれば同様の画面になる。

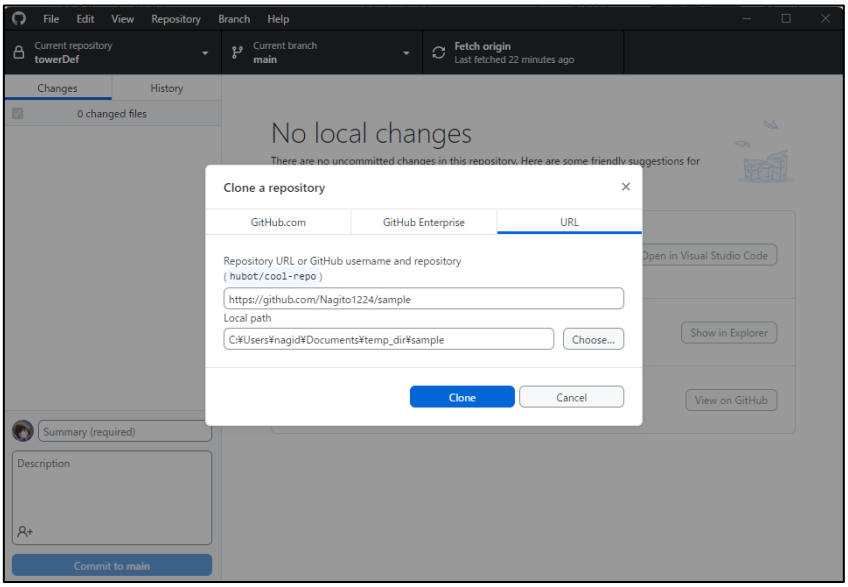

図 9: GitHub Desktop の clone 画面

ここで上にはリモートリポジトリの URL、下にはローカルリポジトリを作りたい任意のディレクトリを入力する。ただし、自動でこの画面に遷移した場合は上の欄はデフォルトで入力されている。

次に、”Clone”を押すとローカルリポジトリが作成され、ワークツリーの中にリモートリポジトリの中身と同じものが作成される。

6 章のプロジェクトの初期化を行っている場合は、ここで 6 章に戻る。

最後に、Unity Hub の”追加”からクローンしたディレクトリ(ワークツリー)を開けば編集を開始できる。

8　編集方法（実践）

8.1　add・commit（ローカル内での変更の保存）

8.2　push（リモートへの反映）

8.3　pull（最新状態のコピー）

途中だけど一旦ここまでの内容をあげておきます