1.7 共同開発の手順

　一つのプロジェクト（開発しているソースファイルやリソースファイルなどの管理単位）を複数人で開発するとき、バージョン管理システムの利用がたいへん有効です。

　以下の図のようにA君とB君で中央リポジトリを使用している場合、A君は自分が行った変更履歴を自分のローカルリポジトリから中央リポジトリへプッシュ（登録）します。そして、B君は中央リポジトリから変更点をプル（取得）することでA君によって変更された部分（差分）を自分のローカルリポジトリへ適用してから自分の開発を行うことができます。

C:\Users\eik2\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\4KGRH4MV\MC900428945[1].wmfC:\Users\eik2\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.IE5\4KGRH4MV\MC900428945[1].wmf

ローカル

リポジトリ

ローカル

リポジトリ

中央

リポジトリ

プル

（変更点の取得）

プッシュ

（変更点の登録）

Aの修正を適用

ソースを修正

A君

B君

　さらに、A君とB君でそれぞれ独自に同じプロジェクトの部分を開発してローカルコミットした場合でも、バージョン管理システムはそれぞれの変更点を重ね合わせる形で新しい変更後のものとして変更履歴をローカルリポジトリとして用意してくれます。この機能をマージ（merge）と呼びます。

　A君とB君がそれぞれまったく別々の部分を修正した場合には、一般にバージョン管理システムが自動的に変更点の重ね合わせをしてくれます（これをオートマージと呼びます）。

　しかし、両者が競合するような変更、例えば二人とも同一の行を変更したなどの場合にはシステムが自動的に重ね合わせ処理を行うことができません。そこでそのような場合には、バージョン管理システムが競合箇所を示して手動による競合解消を開発者に促すような仕組みになっています。

1.8 ブランチ（Branch）

　ひとつのプロジェクトにおいて複数の流れで同時並行的に開発を行う必要が生じることが様々な理由で起こりえます。この要求に対してバージョン管理システムでは「ブランチ（Branch）」という機能で対応することができるようになっています。

　ブランチが必要になる理由としては以下のようなものがあります。

・安定版と機能追加版の同時開発（とリリース）

・複数のグループで意欲的な機能をそれぞれ追加したバージョンを同時開発

・複数のグループでそれぞれ異なった部分の機能向上したものを同時開発　など

安定版（Stable）

の開発

機能追加版1

の開発

機能追加版2

の開発

　Gitでは最初に作られる流れをmasterブランチと呼びます。masterブランチから必要に応じて複数のブランチが作成され、それぞれのコミット（変更履歴）が記録されていきます。

　複数のブランチを統合して一つのブランチにすることや、機能変更などが行われたブランチでの変更をmasterブランチに反映させたりなどといったことが自由に行えます。

　各開発者は複数のブランチを切り替えながらそれぞれのブランチにおける開発（変更）を行っていくことができます。ブランチを切り替えた場合、HEADという名前で指し示されるコミットもそのブランチでの最新コミットへと切り替わります。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **【演習：gitコマンド】**  　「GitHub for Windows」はGUI操作でGitとGitHubを取り扱うことのできる便利なツールでしたが、すべてのgitの機能を自由に使うためにはコンソール画面でCUI操作することが必要になります。  　すべての機能を使うことはgitの詳細を知らない限り難しいですが、ここでは最低限のCUI操作を演習で試してみましょう。  1. CUI操作環境（シェル）を起動する  (1)「Git Shell」を実行  　（Powershell.exeというGitHub for Windowsインストールによって用意されたCUI操作  　　環境が実行されます）  ※CygwinというものをインストールするとUNIXと同様のCUI操作環境が手に入ります）  2. gitコマンドのヘルプを確認する  (1)『git --version』と入力してEnter　（gitのバージョンが確認できます）  (1)『git --help』でEnter　（gitで使える命令の一覧が表示されます）  (2)『git help init』でEnter　（init命令のヘルプが表示されます）  3.ワーキングツリー（作業フォルダ）を作成し、gitで管理を開始する  (1)『mkdir sample2』でEnter　（mkdirは新規フォルダを作成するOSの命令です）  (2)『cd sample2』でEnter　（cdはフォルダ移動するOSの命令です）  (3)『pwd』でEnter　（pwdは今どこのフォルダにいるかを表示するOSの命令です）  (4)『git init』でEnter  　（sample2フォルダがワーキングツリーとして初期化されました。[master]に注目）  (5)『start ,』でEnter　（start .は今いるフォルダをexplorer表示するOSの命令です）  　（.gitという隠しフォルダが自動的に作成されている事を確認してください）  4. gitに管理させるファイルをひとつ用意する  (1)TeraPadなどのテキストエディタで「sampletext2.txt」というファイルを作成  　ファイルの内容は以下の通りにする   |  | | --- | | これはgitに管理させるファイルです。  gitをCUIのコマンドで操作する練習用です。 |   ※必ず文字コードをUTF-8にして保存してください  5. sampletext2.txtの変更点をgitに知らせる（インデックスに追加する）  (1)『git add sampletext2.txt』でEnter  　（[master +1 ~0 -0]に注目）  ※インデックスにsampletext2.txtが登録されてコミットの管理対象となりました  6. ローカルリポジトリにコミットする  (1)『git commit -m "sampletext2.txtを新規作成"』でEnter  　（[master]に戻ったことに注目）  7. コミットのログ（履歴）を確認  (1)『git log』でEnter　（詳細に表示します）  (2)『git log --oneline』でEnter　（1つのコミットあたり1行で簡略表示します）  (3)『git show』でEnter　（最新の変更点について詳細に表示します）  ※暗い黄色の文字で表示されている数字と文字の列が「コミットID」です  8. sampletext2.txtに変更を加える  (1)「sampletext2.txt」をテキストエディタで開き、3行目として次の文章を追加し保存   |  | | --- | | 3行目にこの1行を加えました。 |   (2)何も入力しないでEnter  　（[master +0 ~1 -0]と赤い文字に注目）  9. 変更点をインデックスに登録する  (1)『git add sampletext2.txt』でEnter  　（[master +1 ~1 -0]と緑の文字に注目）  10. インデックスに従って新たにコミット（2回目のコミット）する  (1)『git commit -m "3行目を追加"』でEnter  　（[master]に注目）  11. コミットのログ（履歴）を確認  (1)『git log』でEnter）  (2)『git log --oneline』でEnter  (3)『git show』でEnter  ※このような形で次々と変更点をコミットしていくことで、バージョン管理させることができます。  12. 2つ目のファイルを作成し、インデックスへの登録とコミットを行う  (1)テキストエディタで「futatume.txt」を作成。内容は以下の通り   |  | | --- | | 2つ目のファイルです。  まとめて管理してもらいましょう。 |   (2)『git add futatume.txt』  (3)『git commit -m "futatume.txtを新規作成"』  (4)『git log』『git log --oneline』  13. ワーキングツリーに変更が無かったときにコミットするとどうなるか確認する  (1)『git commit』  ※「作業フォルダに変更がないからコミットしませんでした（nothing to commit, working directory clean）」とメッセージが表示されます  14. コミットメッセージをテキストエディタで入力する機能を確認する  (1)「futatume.txt」の1行目を削除  (2)『git add futatume.txt』  (3)『git commit』  　（テキストエディタが起動して入力を求められます）  　（1行名にコミットメッセージ、2行目は空行、3行目からコメントを入力します）   |  | | --- | | 1行目を削除しました  futatume.txtは1行だけのファイルになりました。  sampletext2.txtに変更はありません。  # Please enter the commit message for your changes. Lines starting  # with '#' will be ignored, and an empty message aborts the commit.  # On branch master  # Changes to be committed:  # (use "git reset HEAD <file>..." to unstage)  #  # modified: futatume.txt  # |   　（テキストエディタをUTF-8で保存して終了させるとコミットが完了します）  (4)『git log』『git show』  　（：と表示された場合はEnterを、(END)と表示された場合はqを押してください）  15. 新しいブランチを作成する  (1)『git branch newbranch』　（newbrnchという名前のブランチを作成します）  16. ブランチの一覧表示と現在のブランチを確認する  (1)『git branch』　（\*が付いているのがいま選ばれているブランチです）  17. ブランチを切り替える  (1)『git chekcout newbranch』　（masterからnewbranchにブランチを切り替えます）  (2)『git log --oneline』  　（ブランチを行った時点からのコミットIDしか表示されないことに注目）  18. newbranchでの変更を加えてコミットを行う  (1)「futatume.txt」を以下のように文章に変更を加えて保存   |  | | --- | | このファイルも、まとめて管理してもらいましょう。 |   (2)『git add futatume.txt』  (3)『git commit -m "futatume.txtの文章を変更"』  (4)『git log --oneline』  19. ブランチをmasterに戻してfutatume.txtの文章を確認する  (1)『git checkout master』  (2)『git log --oneline』  (3)テキストエディタで「futatume.txt」を開いて文章を確認  　（先ほど加えた変更が無くなっているはずです）  20. masterブランチにnewbranchブランチの変更を反映させる  (1)『git marge newbranch』  (2)『git log --oneline』  (3) テキストエディタで「futatume.txt」を開いて文章を確認  　（newbranchブランチで加えた変更がmasterブランチにも反映されているはずです） |

応用練習

　・隠しフォルダ「.git」の中にあるファイルをテキストエディタで見てみましょう

　・fast-forwardマージという用語について調べてみましょう

　・複数のブランチそれぞれに変更を加えて競合を発生させてマージしてみましょう