GITの利用手順

2018年12月6日 木曜日　　9:12

ディレクトリを作成し、作成したディレクトリへ移動

　　git init ：初期化。このフォルダをGitで管理することを教える。

名前とメールアドレスを設定する。

　　git config—global user.name “hoge”

　　git config —global user.email hoge@hoge.com

任意のファイルを作成

　　git add hoge.txt：変更をGitに教える

変更を記録するようGitに依頼する。

　　git commit -m ‘hoge’　　-uオプションでgit addを省略可能

　　git pushして完了

git diff コミットID コミットID：変更履歴を比較する。

git checkout ID hoge.txt：Gitの履歴より復元

git commit —amend (-m “コメントを修正する”)：直前のコミットを修正する。

git log (—oneline)：変更履歴を参照する。

4つのオブジェクト = .git/objects内のblob, tree, commit, tag

参照記録 = .git /HEAD　　.git/refs内のタグやブランチ

上記がどの状態になっているか、どう変化するか常に意識しておく

Gitのフォルダ構造

2018年12月17日 月曜日　　10:43

HEAD：現在作業中のブランチ名を保存
FETCH_HEAD：fetchしたリモートリポジトリとブランチを保存
ORIG_HEAD：前回のHEADを保存
config：このリポジトリ固有の設定
description：git web で利用し、webブラウザで見たときDescriptin項目に表示
index：git add するときのインデックスの実体

hooks/：フック処理を保存
info/：exclude(除外対象ファイル)　　refs(httpアクセスに必要な情報)git update -server -infoが更新
logs/：reflogの実体(テキストのログファイル)
objects/：リポジトリの実体(blob tree commit tag オブジェクトを保存)
refs/：タグ・ブランチなどを参照情報ごとにファイルで保存

packed-refs/：refs/ をgit repackして集約したファイル
branches/：未使用
COMMIT_EDITMSG：コミットメッセージを保存

Gitの用語

2018年12月14日 金曜日　15:05

作業フォルダ：OSが管理するファイルシステムのデータ保存場所

インデックス：git add時にgit形式のフォーマット情報に変更されたファイル情報の一時保存場所

リポジトリ：Gitの変更履歴の保管場所。コミットするとインデックスの情報を追加

コミット：様々な管理情報が付加された一塊のデータ

コミットする：リポジトリへコミットを追加する。からのフォルダはコミット不可

ワーキングツリー：最新のファイルの状態

タグ：SHA1ハッシュ値にわかりやすい名前をつけて保存したファイル

ブランチ：最新のコミットを追跡する動的なタグ。最新のコミットのSHA-1ハッシュ値に変化

HEAD：常にリポジトリ中(作業中のブランチ)の最新の状態を指す。最後にコミットしたもの
→git checkoutでブランチを書き換えるときは、HEADのブランチ名を書き換えるのを忘れない
　同時に作業フォルダをブランチが指すコミットの内容に変更する
　git symbolic -ref HEAD refs /heads/next

コミット

2018年12月14日 金曜日　15:18

git commit (—amend)

HEADの指すコミットに続く歴史になる。

—amendはHEADのコミットを上書きする。直前のコミットを修正する。

git show：最新のコミットを確認、コミットを指定しないとHEADとみなされる。

git log -p：前回のコミットとの差分を表示

git revert コミットID：コミットを打ち消す修正。viが起動。修正理由を記載する。

viで何もしなくてもコミットされる。

git rebase -iコミットID：指定したコミットIDの次のコミットを修正する。viが起動。

pickなどを修正・削除。

git status：Indexを参照。add、commit、conflictしたファイルなどが確認できる。

git reset —hard HEAD^：直前のコミットの取り消し。HEAD^を任意のコミットIDで指定可能

git reset —hard HEAD：コミット後の変更を全て取り消す。1つ前のコミットの状態に戻る。

—hard：commitもファイルの変更も全て取り消す。

—soft：commitのみ取り消し、ファイルの変更は残す。Index, WorkingTreeはそのまま

git rev-parse HEAD：現在のHEADのコミットIDを取得。

git reset —hard ORIG_HEAD：直前のリセットを取り消す。

git reflog：これまでHEADが辿ってきた履歴を見る。resetで問題があるときはこれで対処

reflogで戻したい状態を見つけたら、git reset [オプション][ハッシュ値]をする。

git reset —mixed HEAD：addを取り消す。git reset HEADでも可能。

git rm：インデックスと作業フォルダから指定したファイルを削除する。

git rm —cached hoge：作業フォルダからは削除されない。削除対象はインデックスのみ。

git my foo bar：fooからbarに変更。インデックスと作業フォルダ両方に同名ファイルの存在必須

git resetなどでブランチの内容を過去のコミットに戻した場合、最新だったコミットはすぐには削除されない。タグ、ブランチから参照できる範囲外にあるため、アクセス手段はSHA-1で直接指定するしかない

→git reflog でハッシュ値を参照。reflogの実体は .git/logsにあるテキストファイル

ブランチ

2018年12月14日 金曜日　16:19

ブランチ：Gitの歴史の流れ。デフォルトではmasterというブランチが作成される。
　　ブランチ名を省略すると作業中のブランチ(HEADが示す)が対象になる。

git pullを省略するとHEADが示すブランチになるので、異なるブランチを扱う場合は注意
　→git configで設定しておく
　branch.master.remote = origin
　branch.master.merge = refs/heads/master

git branch：作業中のブランチを確認する。
git branch hoge：hogeブランチを作成
git checkout hoge：hogeブランチへ移動

リモート追跡ブランチ：外部のリポジトリを取り組む際に作られるブランチ。git branch -rで確認
　→自分のリポジトリの中に外部のリポジトリをコピーし、コピー元へのアクセスに利用する

git pull：git fetch + git merge を実行する複合コマンド
　→fetchで外部リポジトリをコピーし、mergeで指定したブランチを合流

異なるブランチの修正を反映させるときは、反映させたブランチへ移動後、マージする

git log —oneline —graph —decorate —all：ブランチを視覚的に表示

ペアリポジトリ：通常は隠しフォルダ.gitに存在するものを通常フォルダに格納したもの
　　命名規則としてhoge.gitとする。作業ファイルなし、コミット不可
git init —bare —shared：任意のフォルダでGitの管理情報を格納

ファストフォワード：プッシュするブランチに共有リポジトリのコミットが全て含まれている状態
　　プッシュする前に誰かが新たなコミットをgit pushするとnon-fast-forward

Gitのリポジトリはリモートとローカルで2つある。
git fetchはリモートリポジトリで更新された最新情報をローカルリポジトリに持ってくる。
git pullと違い、ファイルが更新されるのではなく、origin/masterが更新される。

ローカルリポジトリにはmaster(ローカルの作業場所と結びついている)とorigin/master(リポジトリと結びついているブランチ)の2つの情報が置かれている。

git fetchでは更新時origin/masterが最新になる。masterには更新が行われていない。
git fetch後、git merge origin/master でローカルファイルが最新状態になる。

ブランチ2

2018年12月14日 金曜日　16:56

リモート追跡ブランチの削除

git branch -d hoge：ローカルブランチを削除
git push origin:hoge：共有リポジトリを削除。共有リポジトリへの変更はpushしか行えない
git push origin foo:bar：fooブランチをリモートのbarブランチへプッシュする。
　　fooを指定しないと、何も指定されていない状態になり削除扱いになる
git remote show origin：リモート上の削除されたブランチがあるか確認する。
git remote prune origin：リモート上の削除されたブランチをローカル消す

ブランチをさかのぼる

masterブランチのまま作業を進めていた場合、まずはコミットしておく。
その状態で、新たにブランチを作成する。(e.g. git branch hoge)
HEADはmasterとhogeの2つを示す。
この状態で、git reset —hard hogeでmasterブランチを任意の位置に戻すことができる。

git stash：現在の作業を保留して、別の作業を行う。
git stash list：保存状態の確認
git stash pop：最新の作業環境を復元。復元した内容はstash listで削除。stash applyで残すことも可能
git stash save ‘Hoge’：メッセージを付加。

git stashしたら、ブランチを作ったり、チェックアウトしたり、リセットしたりと自由な作業をして
最後にgit stash popすれば元の作業状態に戻る

no branch：どのブランチにも所属していない状態。no branchのコミットは参照する目印がないので
　　追跡できなくなる。detached HEAD

タグ

2018年12月14日 金曜日　　17:16

タグ：コミットIDは覚えづらいのでタグを用いる。gitコマンドの引数として、利用可能。
　　タグは不変で常に同じコミットIDを指す。
　　git pull → git tag hoge タグ名「hoge」を設定 → git show で表示可能

git checkout hoge：タグ「hoge」を取り消す
git tag -a hoge：タグに注釈を付加。viが起動。メッセージを書き込める。
git tag -n：タグに付加されたメッセージを表示。-n10で10桁表示
git tag -d hoge：タグの削除
git tag hoge コミットID：過去のコミットIDにタグをつける

タグを共有するには、git push origin hogeで共有可能
git push —tags を使用すると全てのtagをプッシュ可能
git push origin:hoge：「:」を付加することで、共有リポジトリのタグを削除

競合

2018年12月14日 金曜日　　17:30

<<<<<　hoge　=======　hoge hoge >>>>>：=====は区切り。括られた箇所を修正する

競合が発生した際に素早く解決する。
git checkout —ours[—theirs] hoge：上側[下側]を指定して、チェックアウトする。

Gitの設定

2018年12月14日 金曜日　17:34

.gitignore：コミットの除外対象リスト

abc：abcという名のファイル・フォルダを無視。否定は!abc

*.abc：.abcで終わるファイル・フォルダを無視。*は0文字以上

*.[abc]：.a or .b or .c で終わるフォルダを無視

abc/：abcという名のフォルダを無視。末尾の/はフォルダを表現

/abc：プロジェクトフォルダのabcという名のファイル・フォルダを無視

優先順位

1 project/.gitignore

2 project/.git/info/exclude

3 $HOME/.gitexclude

git configの優先順位はオプションなし → —global → —system の順

競合した場合は1つしか有効にならない。

git config -l：Gitの設定を確認。

優先順位	影響範囲	設定ファイルの保存場所
なし	各リポジトリの中だけ	各リポジトリの.git/config
—global	ログインユーザ環境	~/.gitconfig
—system	OS環境全体	/etc/gitconfig　/opt/local/etc/gitconfig /usr/local/git/etc/gitconfig

git config —system —unset user.name：unsetで設定の削除

./usr/local/git/contrib/completion/git -complention.bash：コマンドの補完

リモートログインではログイン先のターミナルが起動せず、日本語の設定が動作しない。

必要な設定を、~/.bashrc、~/.bash_profile に書いておく

echo ‘export LANG = ja_JP.UTF-8’ >> ~/.bashrc

windowsでは set LANG = ja_JP.UTF-8

git config —global.core.quotepath false：日本語ファイル名の文字コード展開を防ぐ

scriptやリッチテキストファイルを管理する

git diffの差分表示はtxtファイルのみ、その他の形式はファイルの属性ルールをプロジェクトの.gitattributesに保存する。

リポジトリの整理

git gcは処理の中でgit repack を実行し、複数のオブジェクトのファイルをpackファイルに圧縮する。*.packが圧縮されたデータ本体で*.idxはpackファイルに素早くアクセスするためのインデックスになっている。

コミットの修正・復元

2018年12月17日 月曜日　　11:03

git checkout ID hoge.txt：Gitの履歴より復元

git rm hoge：インデックスと作業フォルダから指定したファイルを削除
→git commit —amend で修正をコミット → 状況を git status で確認

git commit —amend (-m “コメントを修正する”)：直前のコミットを修正する。

2つ以上前のコミットの修正

git log —oneline でIDを探す。修正したいIDの1つ前のIDをコピー
git rebase -i コピーしたID
viで対象のIDをpickからeditに変更
git commit —amend で修正
git rebase —continue で修正完了