git Git の設定をする

```
git config --global user.name "Kouichi Yoshino"
git config --global user.email "yoshino@manifold.co.jp"
git config --global -l
```

エディタを指定

```
git config --global core. editor emacs
git config --global color. ui auto core. editor
```

リポジトリを作る

```
$ git init
Initialized empty Git repository in /.../myProject/. git/
```

ファイルやディレクトリをステージに入れる…

```
git add ファイル ...
git add ディレクトリ ...
```

変更のあったファイルをステージに入れる…

```
git add -u
```

カレントディレクトリ配下をすべてステージに入れる…

```
git add .
```

作業ツリーとステージの差を出力.

```
git diff
```

作業ツリーと HEAD の差を出力.

```
git diff HEAD
```

ファイルやディレクトリをステージから取り除く

```
git rm ファイル
git rm -r ディレクトリ
```

ステージの内容をコミットする

```
git commit
git commit -m 'README ファイルを作成'
```

HEAD が指すコミットをやり直す

cd

```
git commit --amend
```

ステージと HEAD の差.

```
git diff -staged
```

ステージ内にあるファイルの一覧を表示する。

```
git ls-files
git ls-files -s
```

ステージの内容を破棄する。(内容を HEAD に合わせる。(そふと)

```
git reset
git reset -- ファイル
```

ステージと作業ツリーの内容を HEAD に合わせる。

```
git reset --hard
```

履歴とステージの先端を移動する(最近の履歴をなかったものにする)

```
git reset コミット
```

指定されたコミットが HEAD になる。作業ツリーの内容は変わらない。

ステージと作業ツリーの内容も履歴の先端を移動する

```
git reset --hard コミット
```

履歴の先端を移動する(最近の履歴をなかったものにする)

```
git reset --hard コミット\\
```

ステージと作業ツリーの内容は変わらない。

コミットの情報を表示する

```
git show
git show コミット
```

コミット間の差を表示する

```
git diff コミット1 コミット2
git diff コミット1 コミット2 -- ファイル
```

HEAD^, @^	HEAD の 1 個前
HEAD~2, HEAD^^, @~2, @^^	HEAD の 2 個前

コミットからファイルをチェックアウトする（昔のファイルをリポジトリから取り出す）

```
git checkout コミット-- ファイル
```

同名のファイルがある場合には上書きされるので注意。作業ツリーがどれかのコミットと等しい状態、たとえばコミットしてからやる。そうすれば上書きされても恐くない。

```
$ git checkout HEAD~3 -- hello.txt
```

ない。3 つ前のコミットの hello.txt を取り出す

ブランチ作成する

```
git branch ブランチ名
```

現在のブランチの状況を見る

```
git branch -v
```

ブランチの更新履歴を視覚的に見る（履歴をグラフィカルに表示する）

```
git log --all --graph  <--oneline>
gitk --all
```

ブランチの切り替え(ブランチのチェックアウト)

```
git checkout 切り替え先ブランチ名
```

コミットの切り替え(コミットのチェックアウト)

```
git checkout 切り替え先コミット
git checkout -b 切り替え先ブランチ名 切り替え先コミット
```

マージ

```
git checkout 親ブランチ
git merge 子ブランチ
```

```
$ git merge proc
Auto-merging program. txt
CONFLICT (content): Merge conflict in program.txt
Automatic merge failed; fix conflicts and then commit the
    result.
$ git ls-files -s
100644 66239d092d452d7f54f45a48ca84bce32365d08e 1 program.txt
100644 2e5ed5fdf154db48459da3c49ccef4fe5fb031e4 2 program.txt
100644 a5df3c0bd2e2f43293263975a8671d1eb3e2abfb 3 program.txt
$ git status
# On branch master
# Unmerged paths:
# (use "git add/ rm < file >..." as appropriate … resolution)
#
# both modified: program.txt
#
no changes added to commit (use "git add" and/ or "git commi …
$
```

3方向マージの強制

```
git checkout 親ブランチ
git merge -no-ff 子ブランチ
```

コンフリクト発生時にマージ前の状態(gitmerge 実行前の状態)に戻す

```
git merge --ablord
```

これをやらないと、作業ツリーも、ステージも、.git ディレクトリ内も、マージ途中の状態になっている。

ブランチを削除する.

“

```
git branch -d ブランチ
git branch -D ブランチ
```

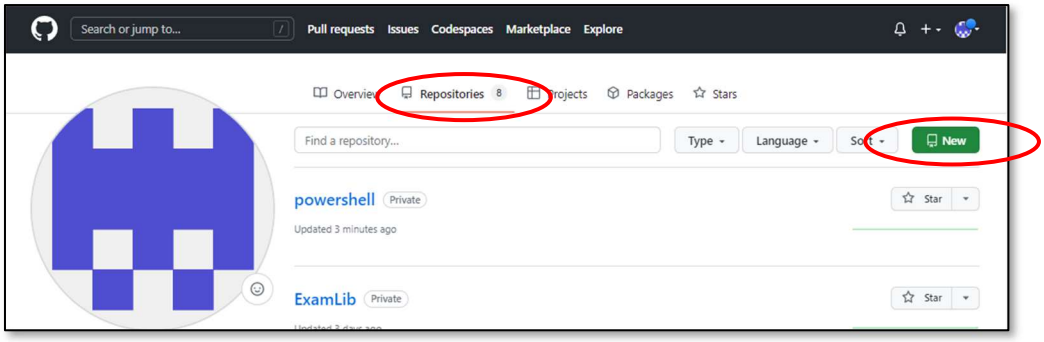
HEAD がいるブランチは、削除できない。未マージのブランチがある場合も、削除できない。ただし、-D ならば削除できる。

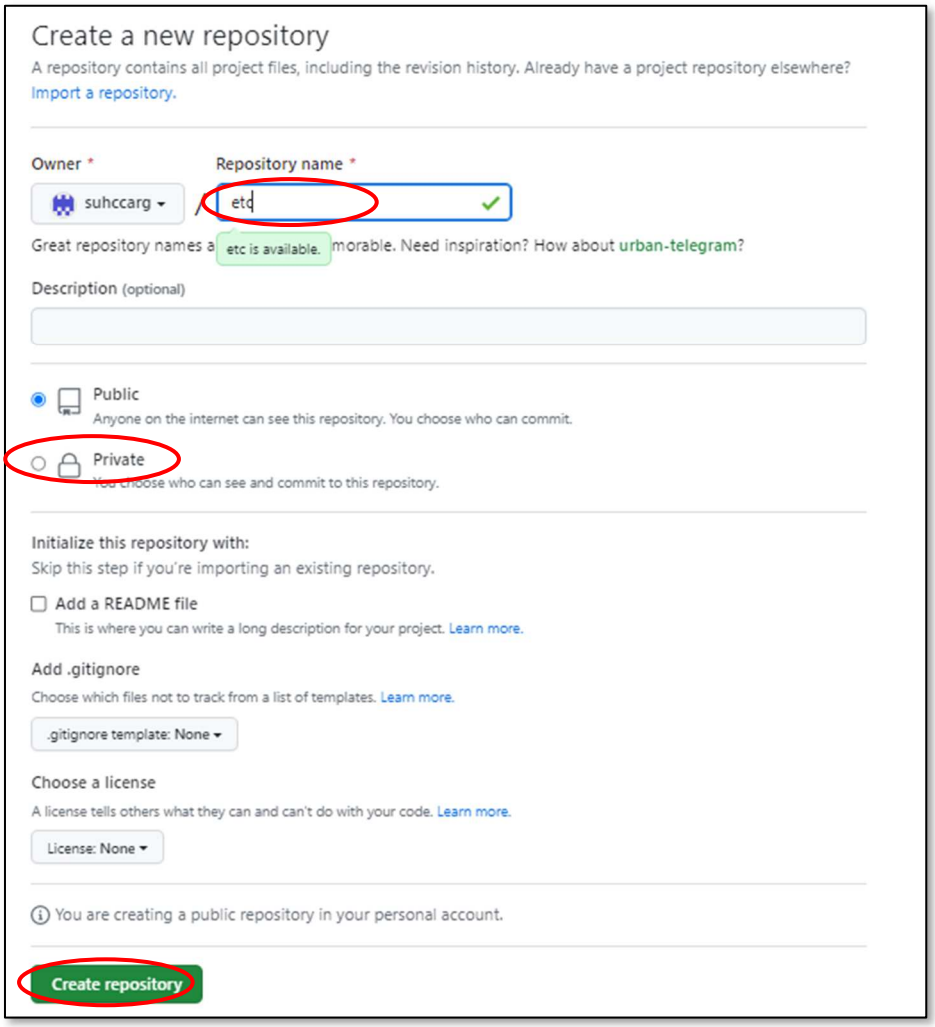
裸のリポジトリ(barerepository)の作成.

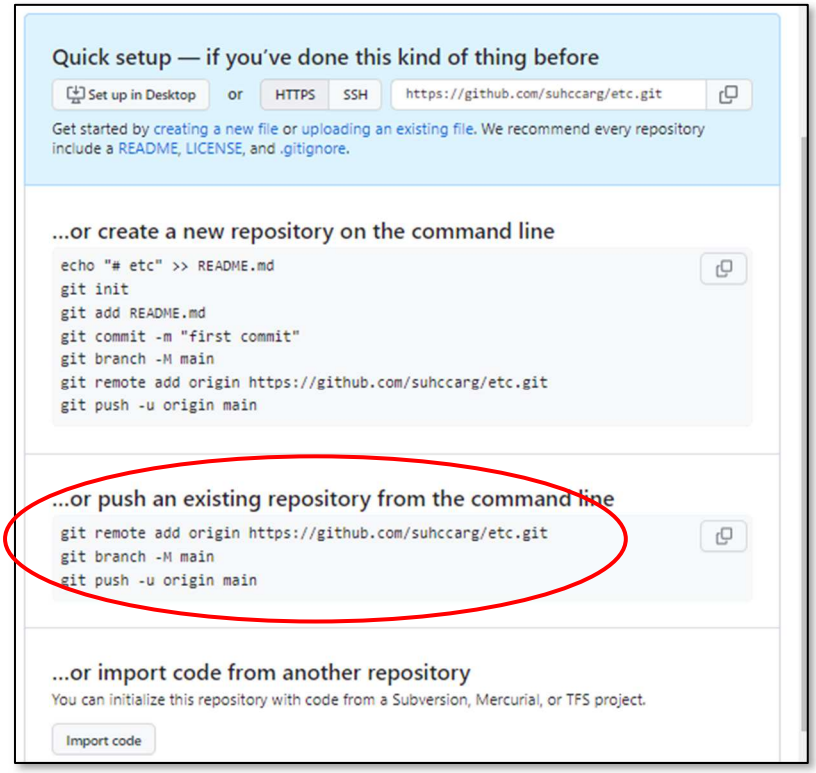
```
git init --bare ディレクトリ 名
```

リモートリポジトリ

リモートリポジトリの利用開始




Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere?
Import a repository.
Owner *
Repository name *
suhccarg
etc
Great repository names a
etc is available.
morable. Need inspiration? How about urban-telegram?
Description (optional)
Public
Anyone on the internet can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with:
Skip this step if you're importing an existing repository.
Add a README file
This is where you can write a long description for your project. Learn more.
Add .gitignore
Choose which files not to track from a list of templates. Learn more.
.gitignore template: None
Choose a license
A license tells others what they can and can't do with your code. Learn more.
License: None
You are creating a public repository in your personal account.
Create repository

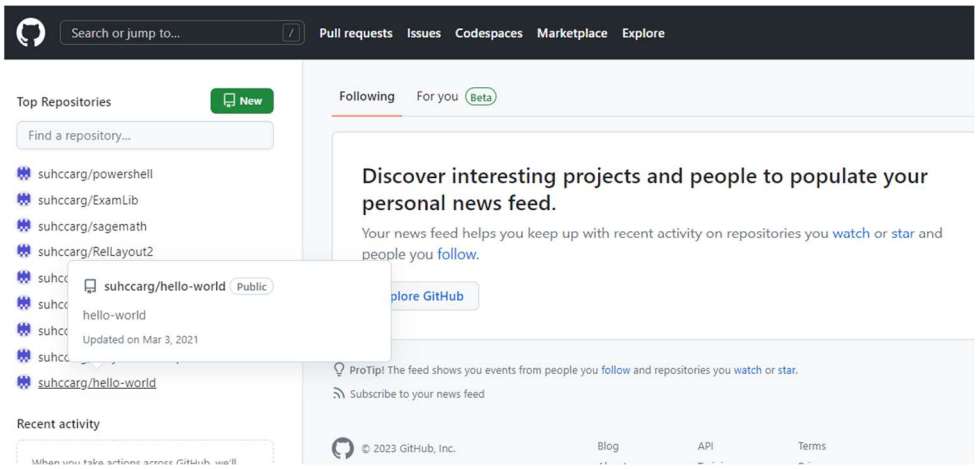



```
$ git remote add origin https://github.com
    jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjs/suhccarg/etc.git
$ git config --global --add safe.directory D:/etc
$ git branch -M main
$ git push -u origin main ←ここで認証
```

-M ブランチ名を強制的に変更（上書き）

-u プッシュすると同時にプッシュ先を上流ブランチにする

リモートリポジトリ削除



リモートリポジトリ add～push

```
$ git add .
$ git commit -m 'hoge hoge'
$ git push  origin main ←ここで認証
```

変更退避

```
$ git stash -u
$ git stash list //退避した作業の一覧を見る
stash@{0}: WIP on test: xxxx
stash@{1}: WIP on commit-sample: xxxx
$ git stash apply stash@{0}  //stash@{0} の作業をもとに戻す
$ git stash apply stash@{0} -index  // staged そのままに
```

コミットしていない変更がある状態で上記のコマンドを実行すると、変更した部分が退避されます。
ワーキングディレクトリ上は差分がない状態になります。
「コミットしていない変更」とは、add したものも add していないものもどちらも含まれます。

タグの削除

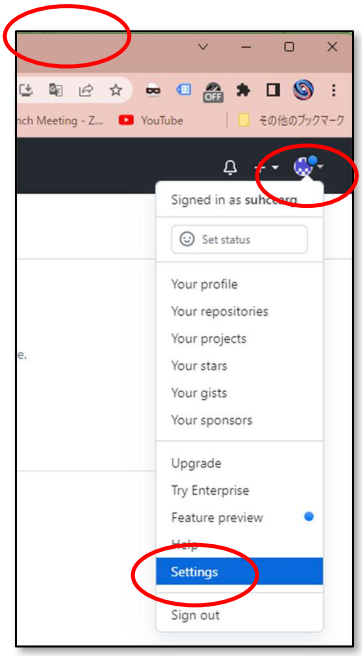
```
$ git tag
$ git tag -d V0.0
$ git tag
```

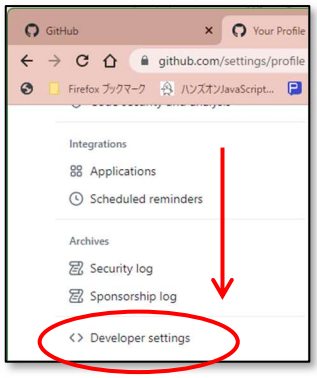
```
$ git ls-remote --tags
$ git push origin -delete V0.0
$ git ls-remote --tags
```

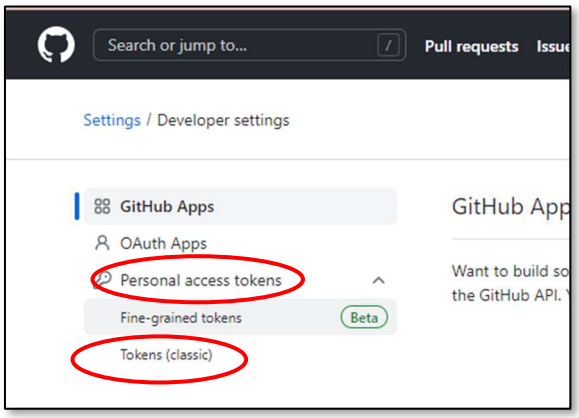
```
git
```

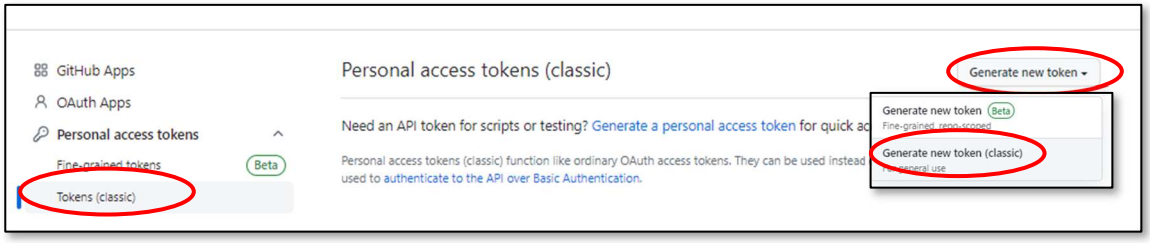
アクセストークン

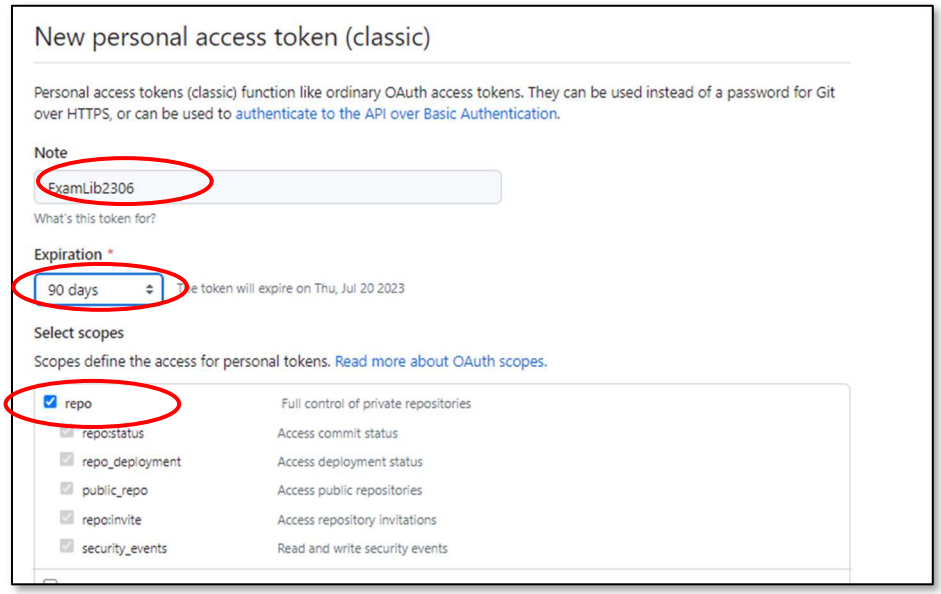
Token の取得

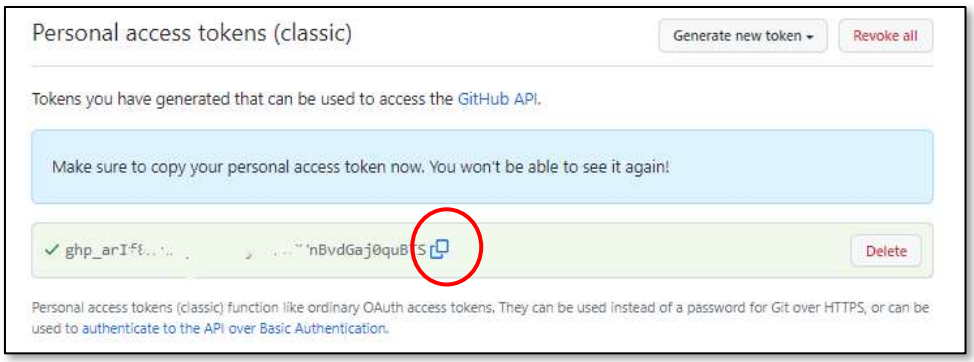













```
$ git config --global credential.helper 'cache --
timeout=7654321'
$ git pull
Username for 'https://github.com': myname
Password for 'https://***@github.com':ghp. . . . . BTS
```

SSH

新しい SSH キーを生成する

$ cd

$ ssh-keygen -t ed25519 -C "your_email@example.com"

PW: type01

SSH キーを ssh-agent に追加する

Git for Windows で ssh-agent を自動的に起動する

bash または Git シェルを開いたときに、ssh-agent を自動的に実行できます。次の行をコピーし、Git シェルの ~/.profile または ~/.bashrc ファイルに貼り付けます。

```
env=~/.ssh/agent.env

agent_load_env () { test -f "$env" && . "$env" >| /dev/null ; }

agent_start () {
    (umask 077; ssh-agent >| "$env")
    . "$env" >| /dev/null ; }

agent_load_env

# agent_run_state: 0=agent running w/ key; 1=agent w/o key; 2=agent not running
agent_run_state=$(ssh-add -l >| /dev/null 2>&1; echo $?)

if [ ! "$SSH_AUTH_SOCK" ] || [ $agent_run_state = 2 ]; then
    agent_start
    ssh-add
elif [ "$SSH_AUTH_SOCK" ] && [ $agent_run_state = 1 ]; then
    ssh-add
fi

unset env
```

```
env=~/.ssh/agent.env

agent_load_env () { test -f "$env" && . "$env" >| /dev/null ; }

agent_start () {
    (umask 077; ssh-agent >| "$env")
    . "$env" >| /dev/null ; }

agent_load_env

# agent_run_state: 0=agent running w/ key; 1=agent w/o key; 2=agent not
    running
agent_run_state=$(ssh-add -l >| /dev/null 2>&1; echo $?)

if [ ! "$SSH_AUTH_SOCK" ] || [ $agent_run_state = 2 ]; then
    agent_start
    ssh-add
elif [ "$SSH_AUTH_SOCK" ] && [ $agent_run_state = 1 ]; then
    ssh-add
fi

unset env
```

最初に Git Bash を実行するとき、パスフレーズを求められます:

```
> Initializing new SSH agent...
> succeeded
> Enter passphrase for /c/Users/YOU/.ssh/id_rsa:
> Identity added: /c/Users/YOU/.ssh/id_rsa (/c/Users/YOU/.ssh/id_rsa)
> Welcome to Git (version 1.6.0.2-preview20080923)
>
> Run 'git help git' to display the help index.
> Run 'git help ' to display help for specific commands.
```