



# GitHubの使い方



# agenda

- ▶ 基本的な用語の説明
- ▶ Githubとは何か、どこがいいのか
- ▶ Gitを使ってみよう！

# 説明に入るその前に・・・

- ▶ GitとGitHubの違いについて簡単に説明すると
- ▶ Git・・・ローカル（自分のパソコン上）でプログラムの変更履歴を管理するためのツール
- ▶ GitHub・・・Gitと連携し、リモート(どこかのサーバ上)でプログラムの変更履歴の管理、公開するツール

基本的には分けて考えなくてよいですが説明上分けて使うので混乱しないように気を付けてください

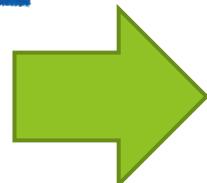
# 基本的用語の説明

- ▶ リモート・・・自分が作業しているパソコンではなくシステム側が提供しているサーバ
- ▶ リポジトリ・・・フォルダとほとんど同じ。ファイルの入れ物
- ▶ アド(add)・・・変更をインデックスに乗せること
- ▶ コミット(commit)・・・変更を保存すること
- ▶ ブランチ(branch)・・・gitで作業する上で履歴を分けて保存するための枝のようなもの
- ▶ マージ(merge)・・・ブランチをくっつける作業
- ▶ リベース(rebase)・・・ブランチをくっつける作業だがマージと方法が若干違う
- ▶ master・・・ブランチのなかで基本となるブランチ、ここでコードを書いたりはしない

飛ばして後で読んでも構いません

# そもそもGitHubとは？

- ▶ 分散型バージョン管理システムの一つ



使う場面から説明していきます！

# トラブルパターン1

- ▶ 一つのプログラムを複数の人で協力して作りたい！

But...どうやって一緒に作ろう？

## 問題点

- ・誰かが作っている時に誰もそのプログラムを触れない
- ・誰かがミスした時にもう取り戻しが付かない

## トラブルパターン2

- ▶ 長くて難しいプログラムを書きたい！

But... どうやって作ろう？

### 問題点

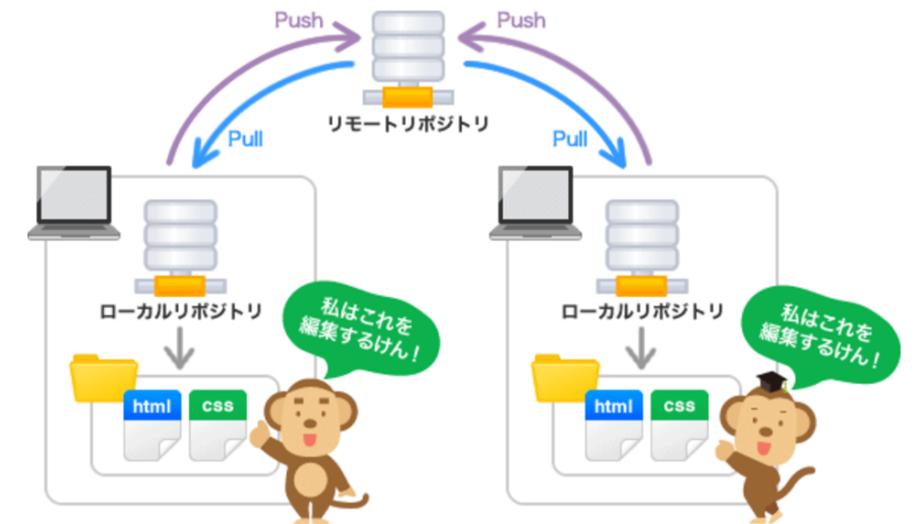
- ▶ あの頃のプログラムの方がよかつた！と思うことがある
- ▶ いちいちコピーを作っていたら面倒くさいしファイルの量が膨大に

# そんな時のためのGitHub!

- ▶ Git, GitHubはローカルやリモートで変更履歴の保存が簡単にできるツールです！
- ▶ 分散型なのでプログラムを作る仲間全員のパソコンで履歴を保存しつつリモートで共有ということができます！



履歴のイメージ



プログラムの共同作成のイメージ

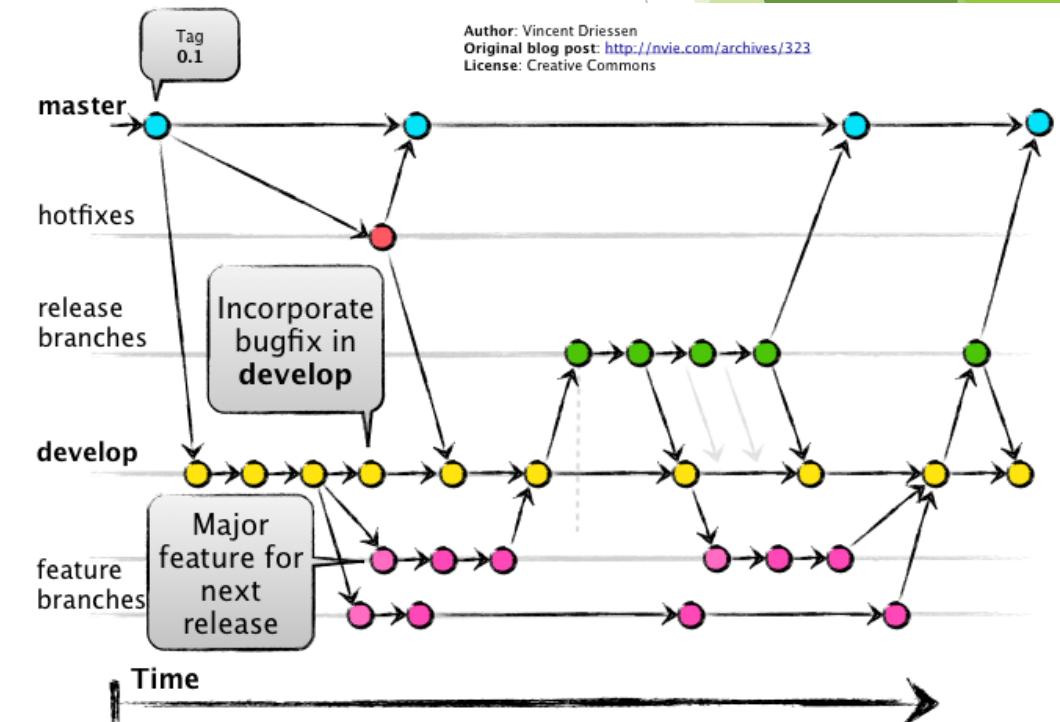


さらに！！！

- ・作ったプログラムを全世界に公開することができるため、それでスキルの証明にもなります！
- ・実際に今回の企業マッチングイベントでは、このGitHubアカウントを企業に見せて評価の基準にしてもらいます！！

# GitHubでの作成のイメージ

- ▶ 具体的には基本の履歴(master)からブランチ(branch)を何本も切って、それぞれのブランチで改善をして良い改善が出来たらくっつけるイメージです
- ▶ この例でいうとmasterに加えhotfixes, release branches, develop, feature branches の四つのブランチがあるイメージです。



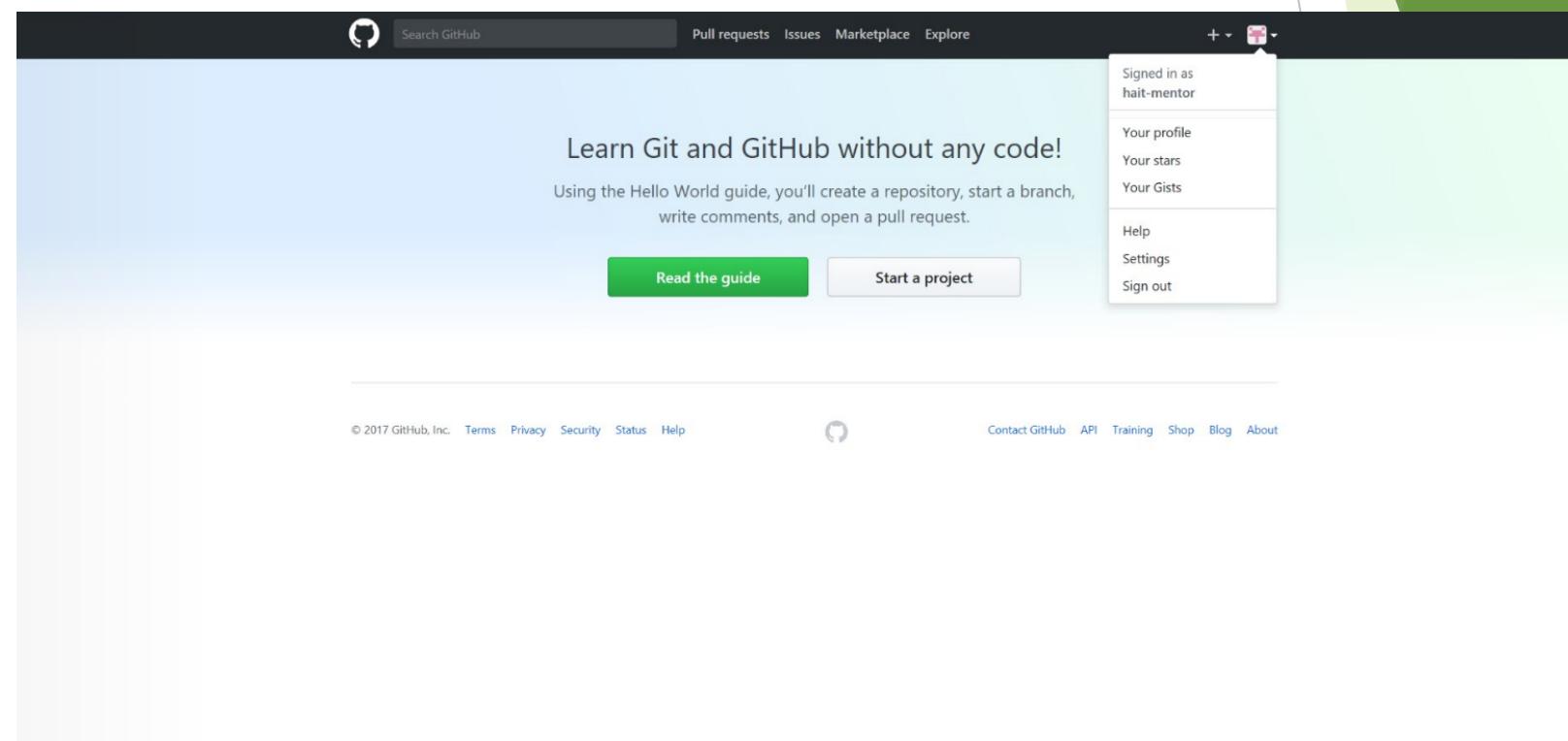
# 実際にGitを使ってみよう(操作に入る前に)

- ▶ まずは操作を行うディレクトリを自分のパソコン上に作ります。
- ▶ Documents上にhaitフォルダを作りましょう。
- ▶ 画像の2つ目のコマンドまでを参考にしてください。

```
C:\$Users\$User>cd Documents  
C:\$Users\$User\$Documents>mkdir hait  
C:\$Users\$User\$Documents>cd hait  
C:\$Users\$User\$Documents\$hait>dir  
    ドライブ C のボリューム ラベルは Windows です  
    ボリューム シリアル番号は 3CC2-4BAD です  
  
C:\$Users\$User\$Documents\$hait のディレクトリ  
  
2017/12/12  18:12    <DIR>          .  
2017/12/12  18:12    <DIR>          ..  
                           0 個のファイル          0 バイト  
                           2 個のディレクトリ   11,472,048,128 バイトの空き領域  
  
C:\$Users\$User\$Documents\$hait>
```

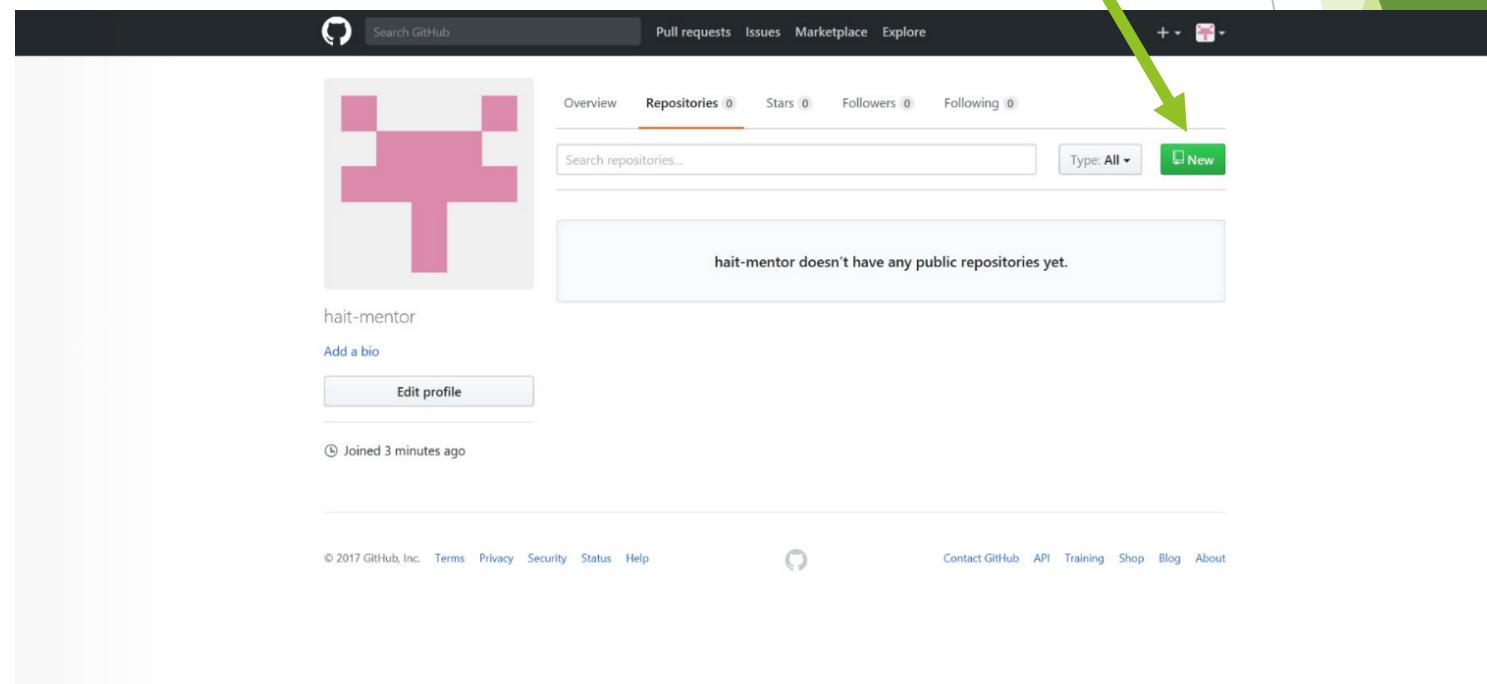
# 実際にGitを使ってみよう (リモートにリポジトリを作成)

- ▶ まずログイン画面から自分のyour profileを開きましょう



# 実際にGitを使ってみよう (リモートにリポジトリを作成)

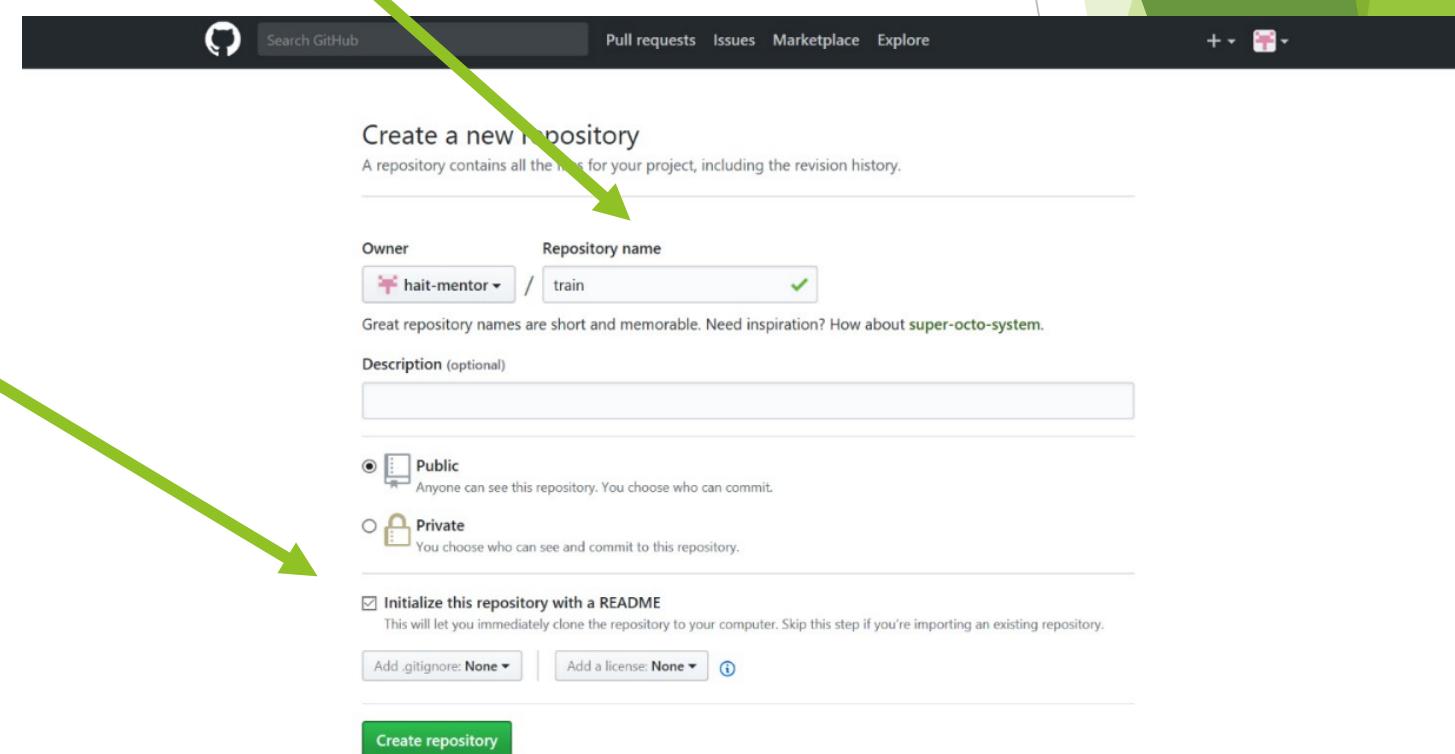
- ▶ 次にRepositoriesを選択し緑のNewを選択しましょう



# 実際にGitを使ってみよう (リモートにリポジトリを作成)

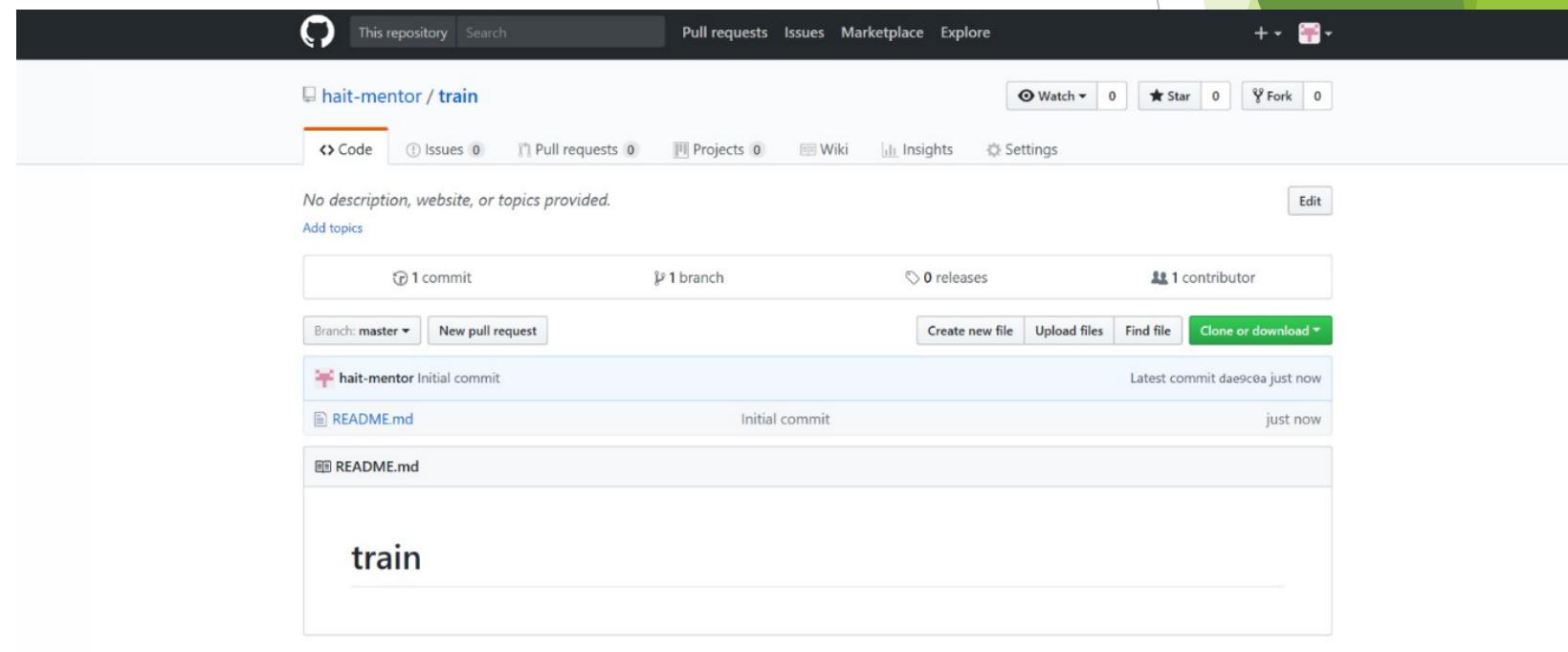
- ▶ 作成画面ではRepository name を仮にtrainにします。
- ▶ Initialize ~ READMEにチェックします

- ▶ Create repositoryを押します



# 実際にGitを使ってみよう (リモートにリポジトリを作成)

- ▶ このような画面になるはずです。これでgithub上にレポジトリを作れました！



# 実際にGitを使ってみよう

- ▶ ここで次の作業のためにclone or downloadを押してリンクをコピーしておきましょう！

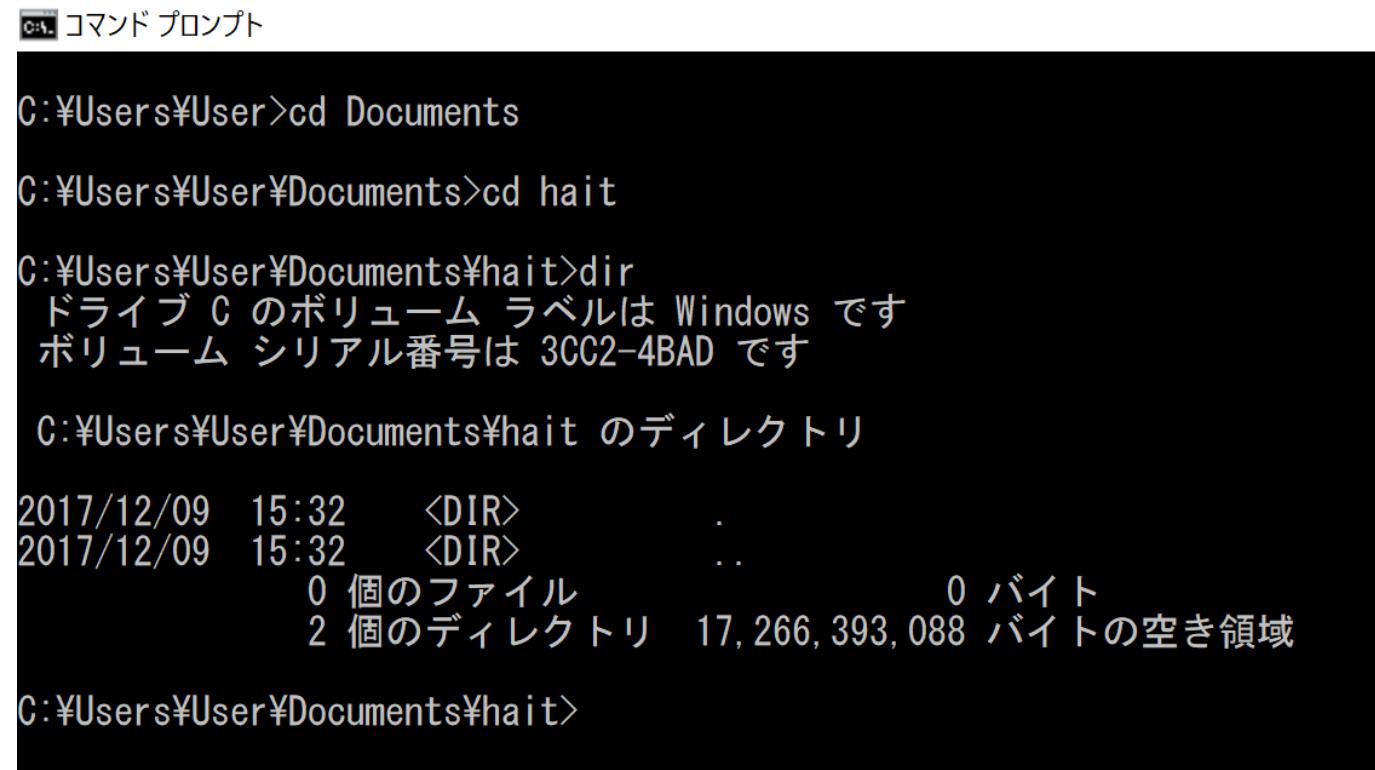
The screenshot shows a GitHub repository page for 'hait-mentor / train'. The page includes a navigation bar with links for This repository, Search, Pull requests, Issues, Marketplace, and Explore. Below the navigation bar, there's a header for the repository 'hait-mentor / train' with icons for Watch (0), Star (0), Fork (0), and Edit. The main content area displays basic repository statistics: 1 commit, 1 branch, 0 releases, and 1 contributor. A 'Clone or download' button is highlighted in green at the bottom right of the stats. To the left of the stats, there are buttons for Branch: master, New pull request, Create new file, Upload files, Find file, and Clone or download. On the right, there are links for Clone with HTTPS (with a URL: https://github.com/hait-mentor/train.git) and Use SSH. Below these are links for Open in Desktop and Download ZIP. The repository itself contains a single file named README.md with the content 'Initial commit'. At the very bottom of the page, there are links for Contact GitHub, API, Training, Shop, Blog, and About.

# 実際にGitを使ってみよう(clone)

- ▶ まずはgit clone でリポジトリのコピーをしてみましょう！  
(現実のイメージとしてはすでに出来上がっている会社のリポジトリをローカル  
上に持ってくるような感じです)

# 実際にGitを使ってみよう(clone)

- ▶ まずディレクトリを置きたい場所(今回はhait)に移動します
- ▶ 中身を見てみると空な事が分かります。(Winはdir, Macはlsコマンドです。)



```
C:\$Users\$User>cd Documents
C:\$Users\$User\$Documents>cd hait
C:\$Users\$User\$Documents\$hait>dir
  ドライブ C のボリューム ラベルは Windows です
  ボリューム シリアル番号は 3CC2-4BAD です

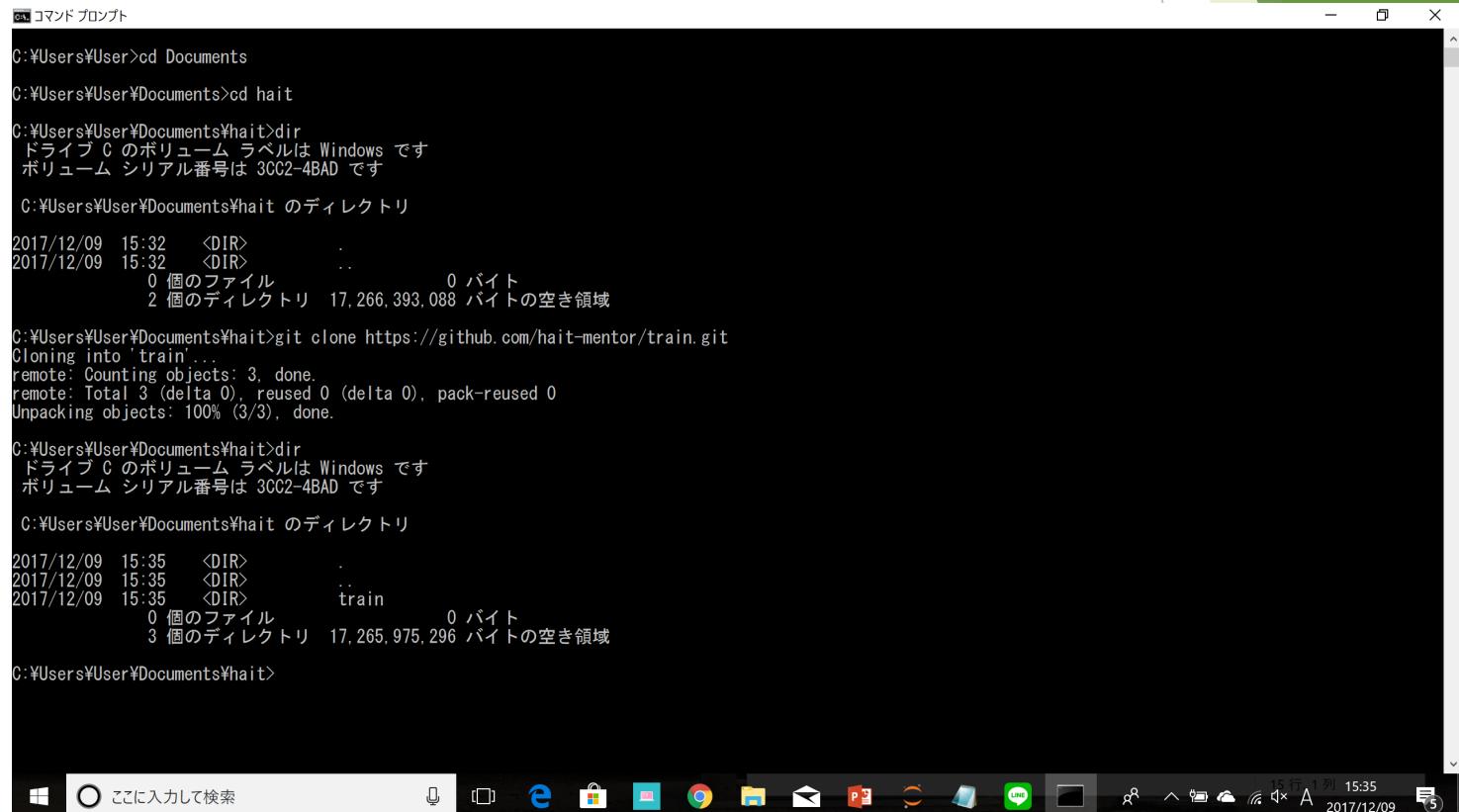
  C:\$Users\$User\$Documents\$hait のディレクトリ

  2017/12/09  15:32    <DIR>   .
  2017/12/09  15:32    <DIR>   ..
                           0 個のファイル          0 バイト
                           2 個のディレクトリ  17,266,393,088 バイトの空き領域

C:\$Users\$User\$Documents\$hait>
```

# 実際にGitを使ってみよう(clone)

- ▶ git clone [コピーしたURL] と入力してみましょう
- ▶ このコマンドを実行してみるとtrainディレクトリが増えていることが分かります。



```
コマンドプロンプト
C:\Users\%User>cd Documents
C:\Users\%User\Documents>cd hait
C:\Users\%User\Documents\hait>dir
  ドライブ C のボリューム ラベルは Windows です
  ボリューム シリアル番号は 3CC2-4BAD です

C:\Users\%User\Documents\hait のディレクトリ

2017/12/09 15:32 <DIR> .
2017/12/09 15:32 <DIR> ..
          0 個のファイル          0 バイト
          2 個のディレクトリ 17,266,393,088 バイトの空き領域

C:\Users\%User\Documents\hait>git clone https://github.com/hait-mentor/train.git
Cloning into 'train'...
remote: Counting objects: 3, done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

C:\Users\%User\Documents\hait>dir
  ドライブ C のボリューム ラベルは Windows です
  ボリューム シリアル番号は 3CC2-4BAD です

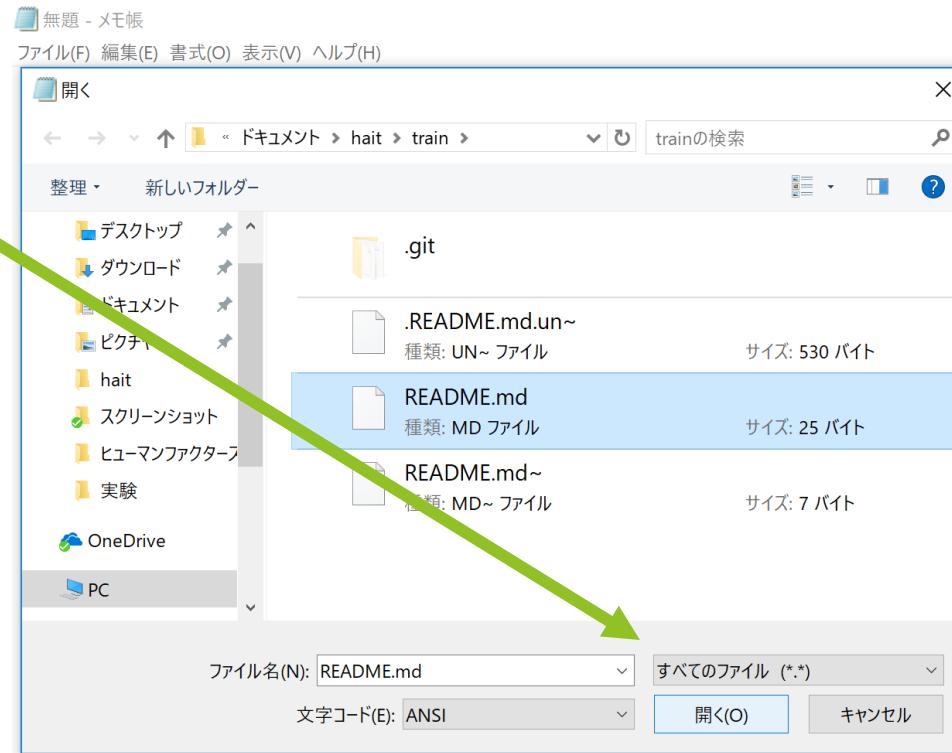
C:\Users\%User\Documents\hait のディレクトリ

2017/12/09 15:35 <DIR> .
2017/12/09 15:35 <DIR> ..
2017/12/09 15:35 <DIR> train
          0 個のファイル          0 バイト
          3 個のディレクトリ 17,265,975,296 バイトの空き領域

C:\Users\%User\Documents\hait>
```

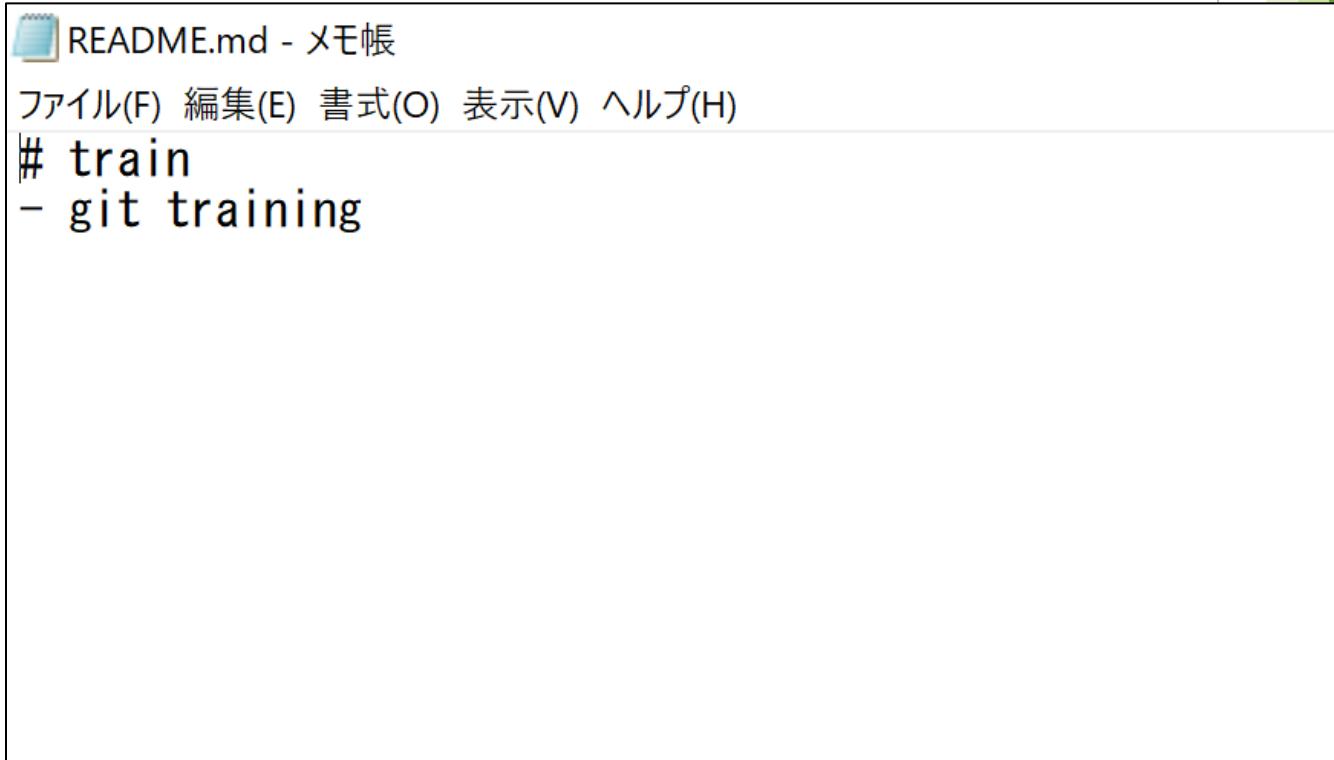
# 実際にGitを使ってみよう！

- ▶ 持ってきたリポジトリの中のREADME.mdを編集してみましょう。中身・手段は何でもいいです。
- ▶ 今回は例としてWinではメモ帳、Macではテキストエディタを使います。
- ▶ メモ帳から選択をすべてのファイルにした上でREADME.mdを編集してみましょう。



# 実際にGitを使ってみよう！

- ▶ 編集の例です。編集の方法、中身は何でも構いません。
- ▶ 保存したらコマンドプロンプトorターミナルに戻りましょう

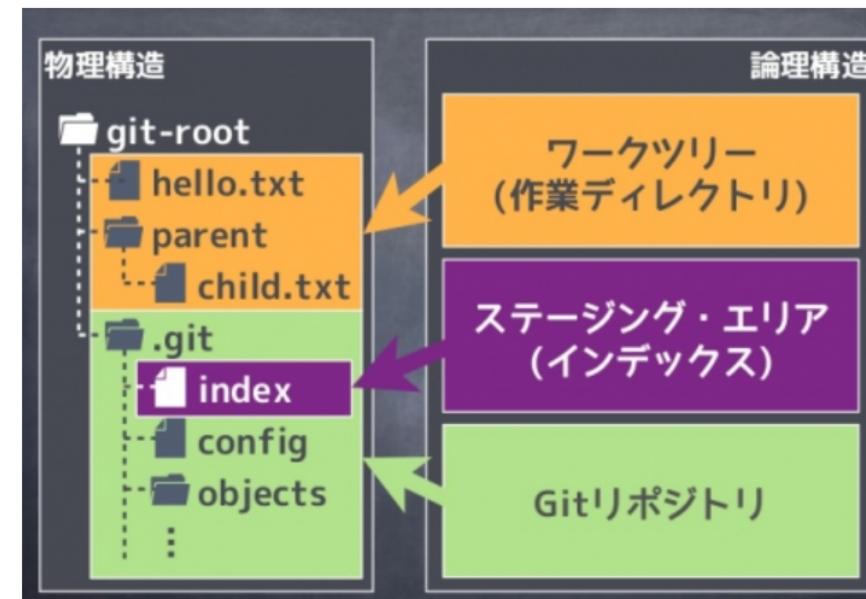


The screenshot shows a text editor window with a white background and a black border. At the top left is a small icon of a document with horizontal lines. To its right is the text "README.md - メモ帳". Below this is a menu bar with Japanese text: "ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)". Underneath the menu is some code-like text:  
# train  
- git training

# 実際にGitを使ってみよう

- ▶ これからがgitの本領であるadd,commit,pushという変更の保存の流れなわけです  
が、ここで今までにない概念で初見では理解が難しいと思うので簡単に  
add,commit,pushの説明をします。
- ▶ まず現時点でのディレクトリの構造は以下のようになっています。
- ▶ hello.txtはいわばREADME.mdのような保存しておきたいファイル

ローカルPCでバージョン管理をしているとき、Gitはこの3つのエリア間で  
データをやりとりしている。



# 実際にGitを使ってみよう(addの前に)

- ▶ 実際にadd,commit,pushはのような流れとなっています。



- ▶ より詳細な説明がほしい人はリンク先のスライドP78~93を参照
- ▶ <http://www.slideshare.net/matsukaz/git-28304397>

# 実際にGitを使ってみよう!

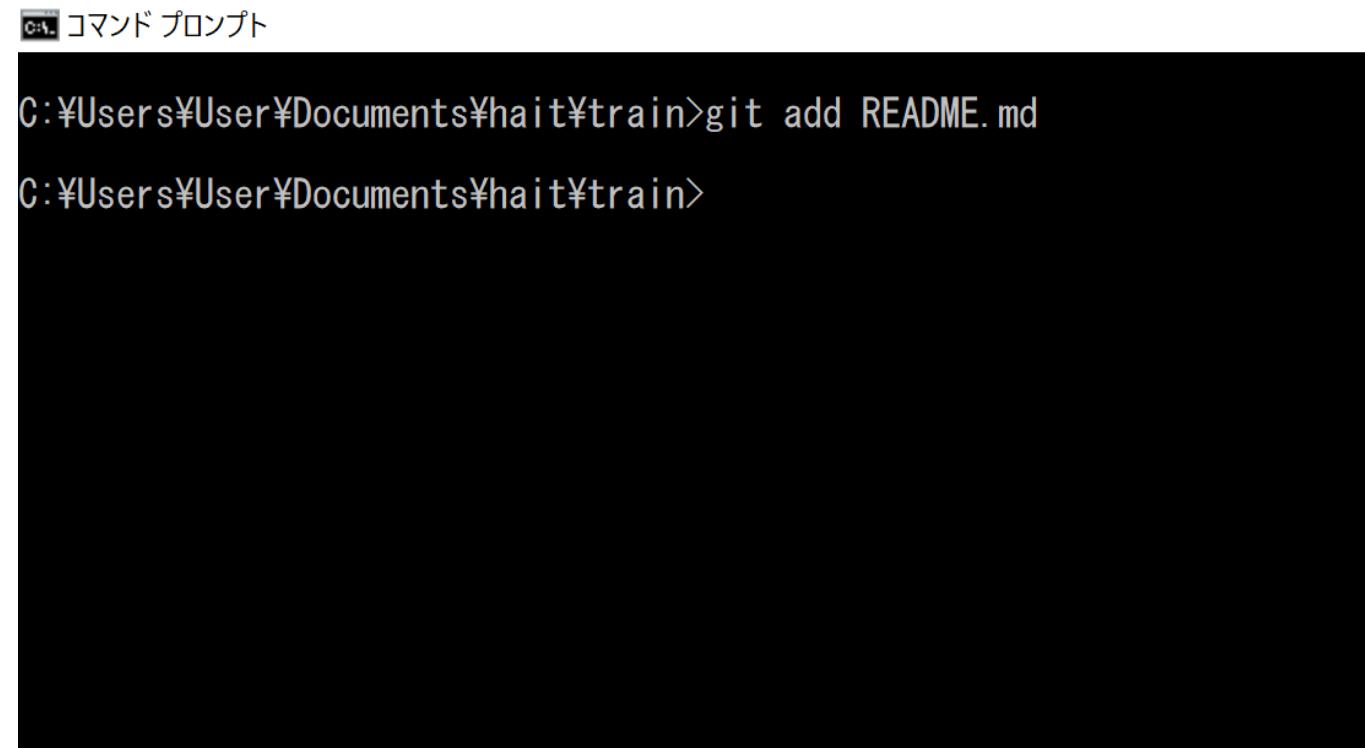
- ▶ 編集が終わったら作業ディレクトリ(train)に進みます。

```
C:\Users\User\Documents\hait>cd train
```

```
C:\Users\User\Documents\hait\train>
```

# 実際にGitを使ってみよう!(add)

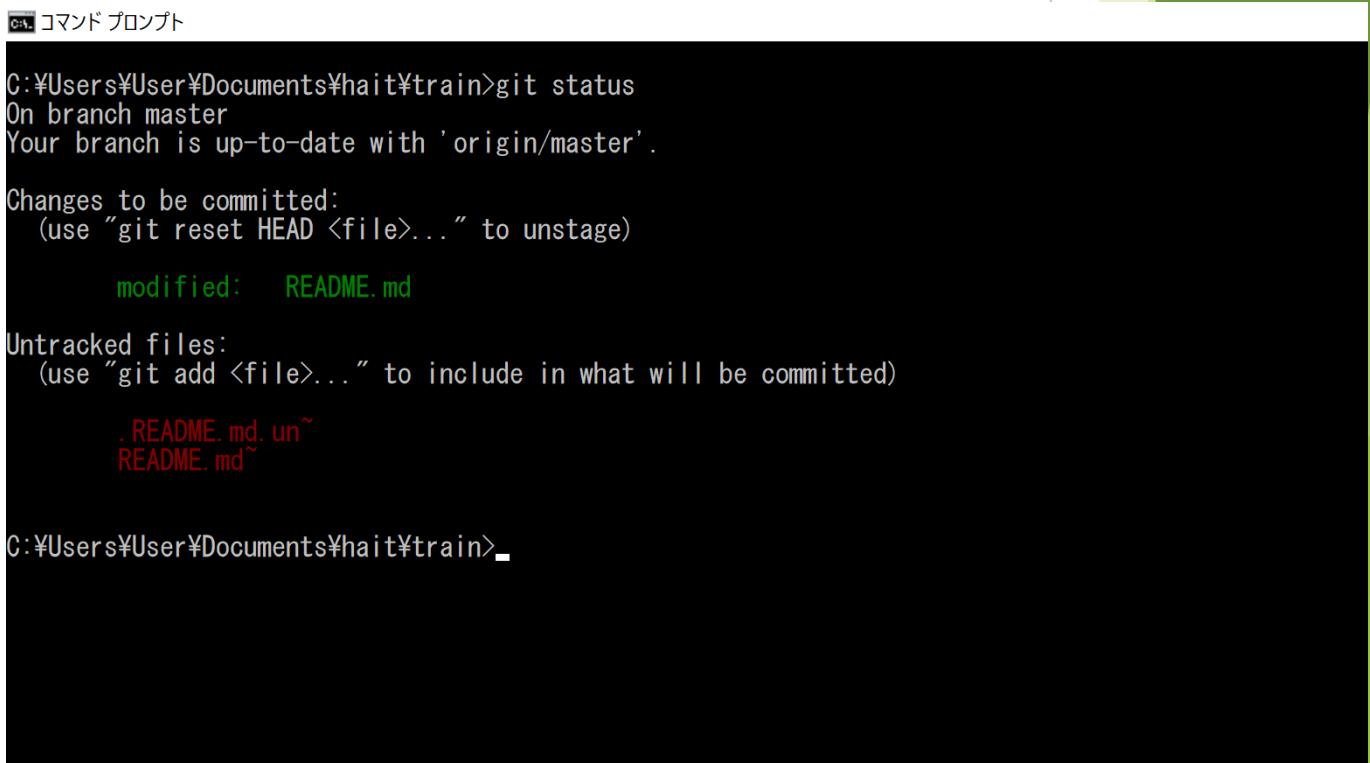
- ▶ 編集済みのファイルをステージングエリアにaddしてみましょう



```
□ コマンド プロンプト
C:\$Users\$User\$Documents\$hait\$train>git add README.md
C:\$Users\$User\$Documents\$hait\$train>
```

# 実際にGitを使ってみよう！(status)

- ▶ Git status でステージと作業ディレクトリの差の状態を見てみると確かに README.mdがmodifiedされていることが書いてあります。



```
PS C:\Users\User\Documents\hait\train>git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   README.md

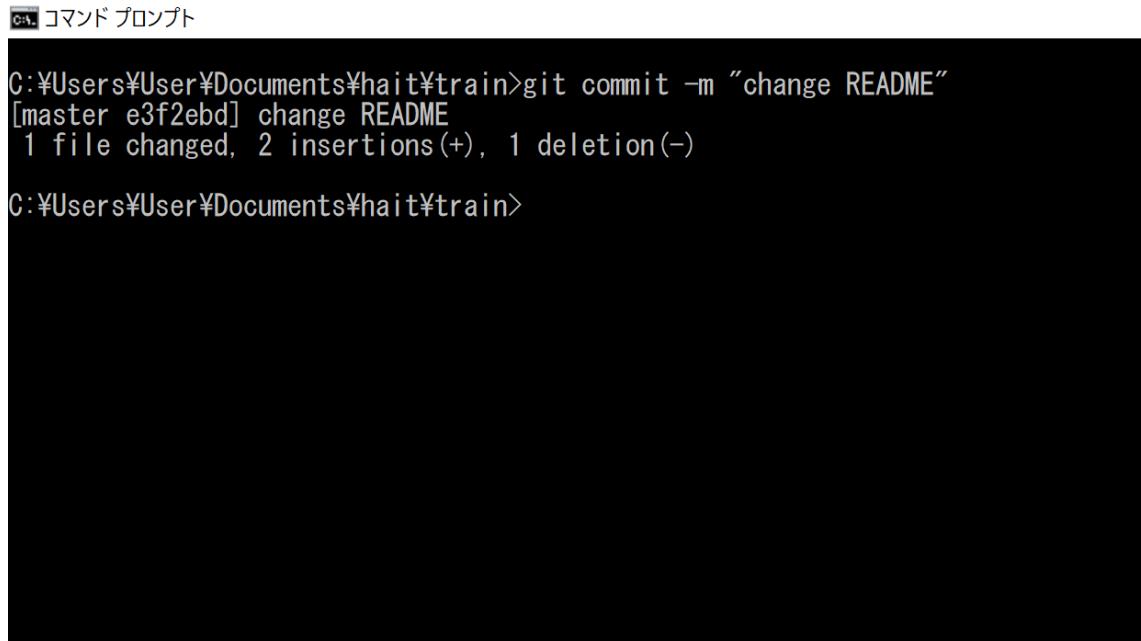
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .README.md.un~
    README.md~

C:\Users\User\Documents\hait\train>
```

# 実際にGitを使ってみよう！(commit)

- ▶ 変更をgit commitでgitディレクトリにあげてみましょう！
- ▶ -m 以下に””で囲んでコメントを入れます。このコメントはgithubでも確認されるので、今回どのようなファイルをコミットしたのかを書いておきましょう。



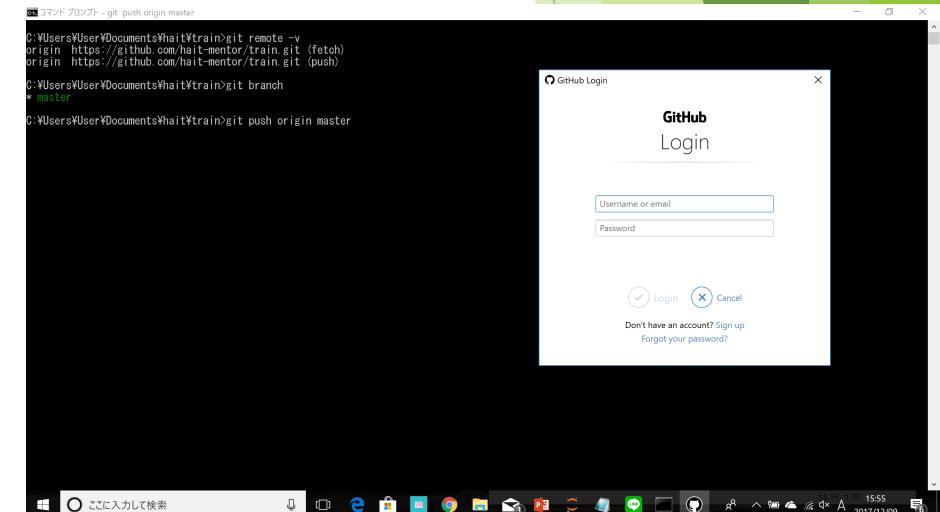
A screenshot of a Windows Command Prompt window titled "コマンド プロンプト". The command entered is "git commit -m "change README"". The output shows the commit was successful, adding a file and changing two lines. The prompt then changes to "C:\Users\User\Documents\hait\train>".

```
C:\Users\User\Documents\hait\train>git commit -m "change README"
[master e3f2ebd] change README
 1 file changed, 2 insertions(+), 1 deletion(-)

C:\Users\User\Documents\hait\train>
```

# 実際にGitを使ってみよう!(push)

- ▶ ステージングエリアに保存した変更をリモートのリポジトリに保存してみましょう
- ▶ まずはgit remote -v で先ほどのリンクがoriginで登録されていることの確認
- ▶ またgit branch で自分が今いるブランチがmasterであることを確認します。
  
- ▶ ではgit push origin master と打ってpushを命じましょう。ここでログインが求められます。
- ▶ Macの方はターミナル上にパスワードの入力が求められますが、セキュリティ保護のため入力しても画面に表示はされないので注意してください。



# 実際にGitを使ってみよう(push)

## ▶ 完了後の画面

```
□ コマンド プロンプト

C:\$Users\$User\$Documents\$hait\$train>git remote -v
origin https://github.com/hait-mentor/train.git (fetch)
origin https://github.com/hait-mentor/train.git (push)

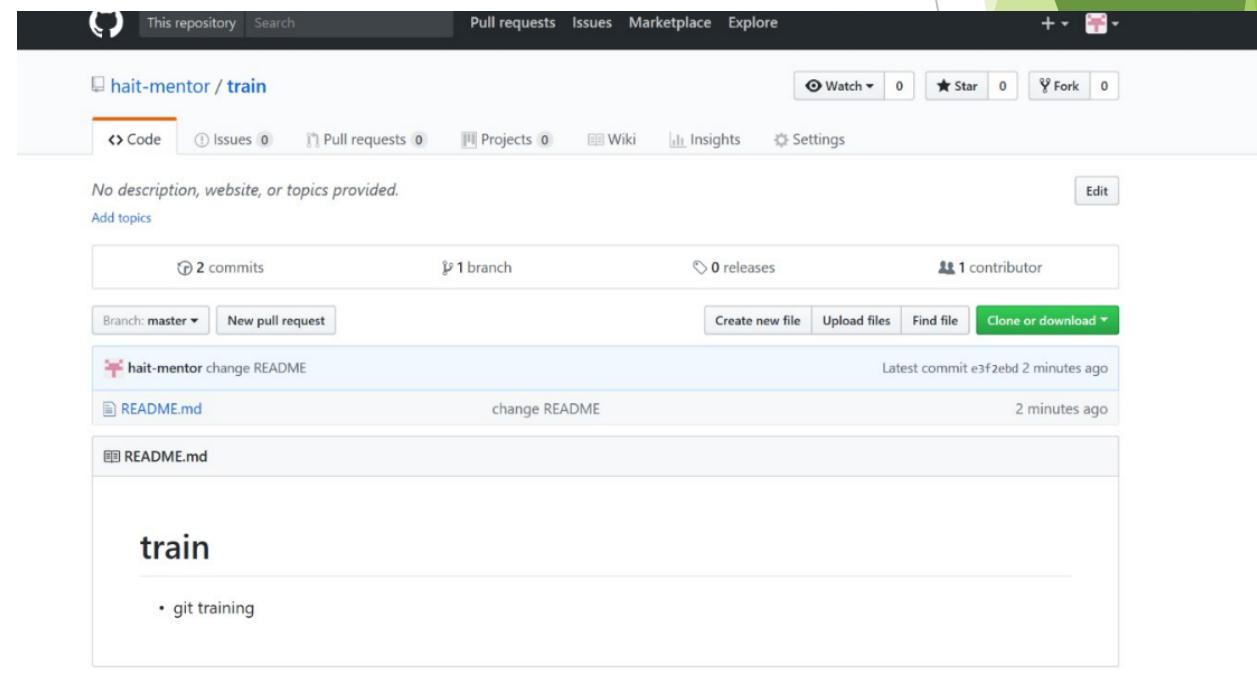
C:\$Users\$User\$Documents\$hait\$train>git branch
* master

C:\$Users\$User\$Documents\$hait\$train>git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 266 bytes | 133.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/hait-mentor/train.git
    dae9c0a..e3f2ebd  master -> master

C:\$Users\$User\$Documents\$hait\$train>
```

# 実際にgitを使ってみよう!

- ▶ 実際にGitHub上で変更が反映されたことを確認してみましょう！
- ▶ git trainingの一文が追加されたことがわかります。

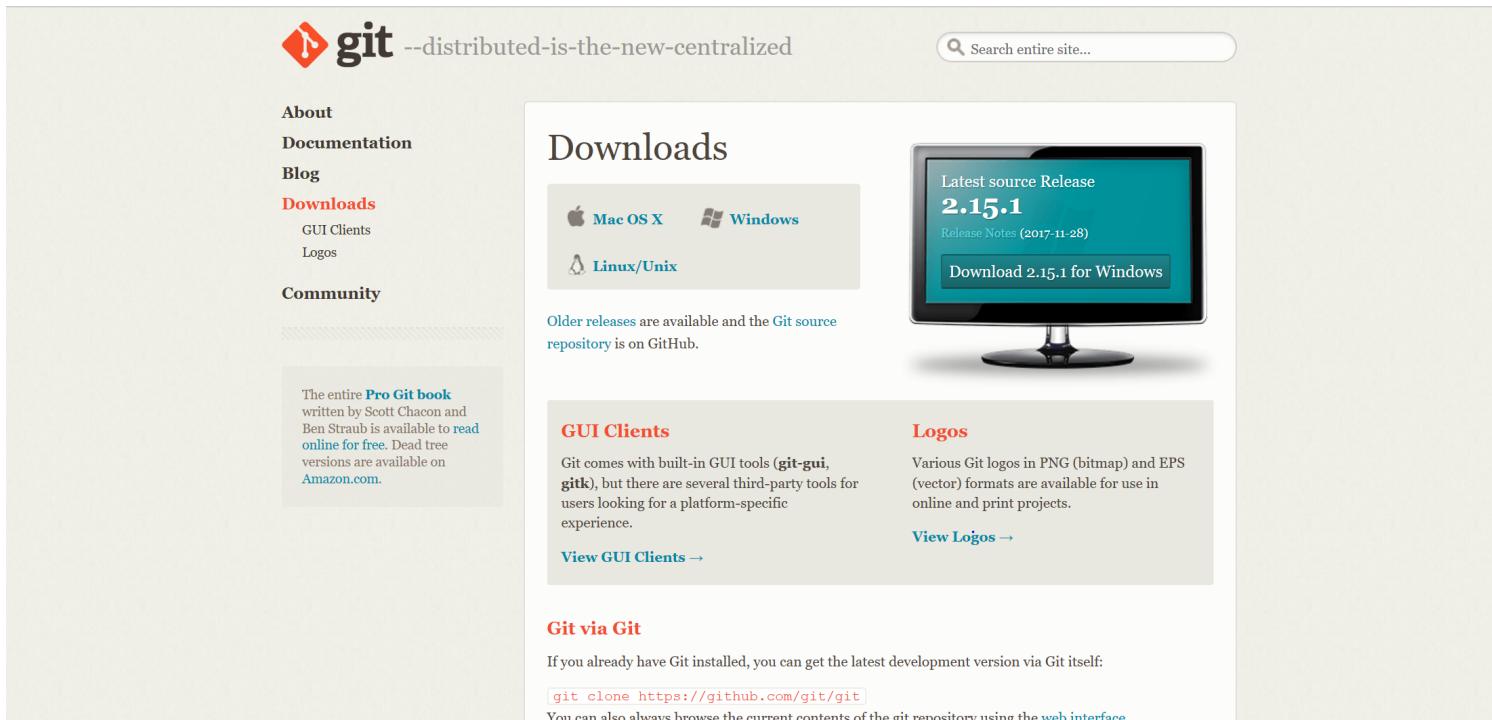


The background features a large, abstract graphic element on the right side. It consists of several overlapping triangles and trapezoids filled with different shades of green. The colors range from a bright lime green on the far right to a darker forest green on the left and top. The shapes are oriented diagonally, creating a sense of depth and movement.

# appendix

# インストール(windows版)

- ▶ まずはgitをインストールしましょう
- ▶ <https://git-scm.com/downloads> よりwindows版をインストール



# インストール(windows版)

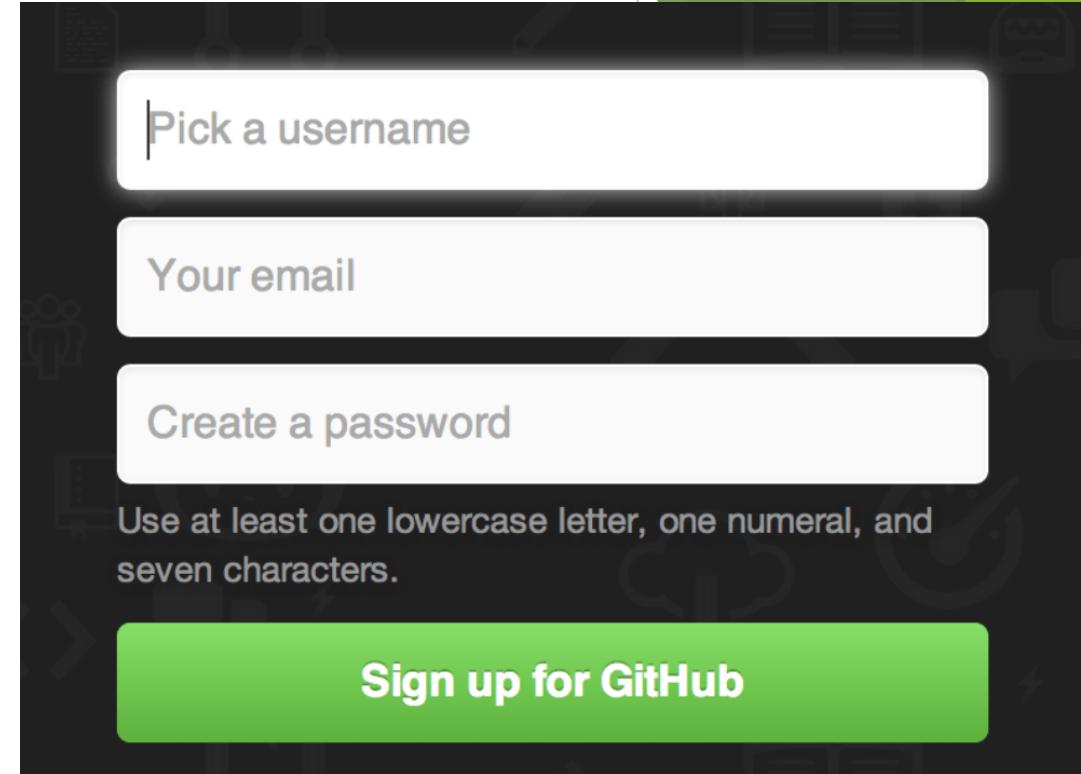
- ▶ <http://www.atmarkit.co.jp/ait/articles/1603/31/news026.html>
- ▶ <http://akiyoko.hatenablog.jp/entry/2014/03/15/032915>
- ▶ こちらを参考に

# 実際にGitを使ってみよう(補足)

- ▶ 今回は最初から設定されているので大丈夫なのですがgit cloneではなく既存のレポジトリをgitで管理したいときはgit initコマンドで管理を始めます。

# Githubアカウントの作成

- ▶ <https://github.com/>
- ▶ こちらを開いてサインアップしてください!



# 実際にGitHubを使ってみよう！補足 (merge)

- ▶ 変更されたブランチをマージしてみましょう。流れ的には

**Pull request**  
(マージしてもらうように  
管理者にお願いする)



**Merge**  
(管理者が良いと思った  
ら許可する)

となると思います。これもインターン前にやっておくいいと思います。

その他にももっと知りたい人へ

- ▶ <https://matome.naver.jp/odai/2136491451473222801>