

OSSを読むススメ

なぜOSSを読むのか？

- コードを書いている時 「**どういうふうにかくのが良いのだろうか？**」 と考えたことは無いでしょうか？
- 書けるんだが **本当にこの書きぶりでいいのだろうか** と思ったことは無いでしょうか？
- 自社のプロダクトコードに **参考になるコードが無い！**？

そんな時こそ OSS を読もう！

OSSを読むことによって解決すること

- こういう時は **どのようにして書いたら良いんだろうか？**
- 一般的には **どのように実装するのだろうか？**
- **なんか書けたけど もっとDryに書けるのではないだろうか？**
- **自社のプロダクトに 参考になるコードがない！？ どうしよう.....**

OSS 最高



ちょっと待って！

Qiita や Zenn の記事では駄目なの？

Qiita や Zenn の記事では駄目なの？

- 記事が書かれた 時期によって参考にすべきかどうか を考えないといけません
- アプリケーションは日々進化している ため、数年前の記事はもう使えないものになっている かもしれません
- OSSのコードは日々進化していて 最新を追従していることが圧倒的に多い
- なんだかんだ言って 公式ドキュメントが最強 である

いい記事もいっぱいあるので

自分で 取捨選択して

Qiita や Zenn の記事も参考にしてね👁👁

例えばこの人の記事は とても参考になります



どどんき
@dodonki1223

🔗 🐦 📧

4189 Contributions

20 投稿	68 フォロー	300 フォローワー
----------	------------	---------------

Web界隈でひっそりと暮らしているソフトウェアエンジニアをこよなく愛するクマの顔のです 大学システム/5年/VB.NET/E C/1年/PHP/広告システム/3年/Rails,Go,Vue.js/coming soon 雑魚エンジニアは雑魚なりにがんばります

プロフィールを編集する

フォロー中のタグ

フォロー中のタグはありません

Organizations



\$ analyze @dodonki1223

投稿した記事	LGTMした記事	回答した質問
Ruby: 35%	JavaScript: 11%	No data
JavaScript: 30%	AWS: 11%	
CircleCI: 25%	CircleCI: 9%	
Bash: 15%	初心者: 9%	
SQL: 15%	Rails: 8%	

ピックアップ記事



@dodonki1223 (どどんき) · CircleCIプロガーコミュニティ
2021年12月15日

3年の運用で編み出した CircleCI 超設計大全

🔗 Ruby, RSpec, CircleCI, Slack

👤 86 🗨️ 1



@dodonki1223 (どどんき)
2021年12月09日

ググリカ、それはエンジニアには必須の能力である

🔗 Ruby, JavaScript, Bash, Slack, VSCode

👤 1416 🗨️ 6 +6人



@dodonki1223 (どどんき)
2019年01月28日

まだトラックパッドでChromeを操作しているの？ そろそろキーボードだけで操作しようぜ！！ Chromeを使い倒そう！！

🔗 Vim, JavaScript, Chrome, bookmarklet, chrome-extension

👤 911 🗨️ 13 +13人

自分が参考にする記事

1. 公式ドキュメント

2. OSS

3. 英語の記事

4. Qiita、Zenn

皆さんは **何を参考** にしますか？

OSS の読み方

- 1つのプロジェクトだけでなく 複数のプロジェクトを参考にする
- OSS同士で 似たような書きぶり があるのでそれを参考にする
- 中身を理解しようとするのではなく 書きぶりを見る

なぜ書きぶりを見るの？

OSSを読んでいて自分はよくあることなのですが **読んでも理解できないことがあります**😭

これは完全に自分の実力不足もあるんですが **処理をする時の前提条件などはOSSで把握するには時間がかかります.....**

なので中身を理解しようとせずに **書きぶりに注力します！**

なぜ書きぶりを見るの？

書きぶりを元にGoogleで検索することで 理解が深まります

なんだこのメソッドは🤔



Google検索



おお！そんなものがあったのか😱

みたいな気付きがよくありました！

中身を理解できる方は理解した上で読むと良いと思います！

実際にRailsのOSSを読んでみる

2022/04/25 時点の情報ですので古い可能性があります。

リポジトリ	Rails	Ruby	star
forem	7.0.2.3	3.0.2	18,982
gitlabhq	6.1.4.7	2.7.4	22,890
huginn	6.0.4.4	2.6.5	35,447
postal	5.2.6.2	2.6.9	11,542
mastodon	6.1.5	3.0.3	27,613
discourse	最新	2.7.2以上	33,541
sprees	6.1 or 6.0 or 5.2	2.5 or 2.6 or 2.7 or 3.0	11,784

before_destroy を調べてみる

以前、ある機能改修でのことです
後輩に **before_destroy** を使用しましたと報告を受けました

ただ自分が **before_destroy** を使用したことがなかったのでこの
Active Record コールバック は一体どういう時に使うべきなのだろう
か.....と思い調べてみました

単純に検索してみる

Googleで検索してみるとTOP
には model の削除可否チェッ
クが出てきています

The screenshot shows a Google search for "before_destroy". The search bar at the top contains the text "before_destroy". Below the search bar, there are navigation links: "すべて", "地図", "動画", "ショッピング", "ニュース", "もっと見る", "設定", and "ツール". The search results are displayed below, showing approximately 69,100 results in 0.53 seconds.

The first result is from <https://qiita.com> with the title "【Rails】 before_destroyでmodelの削除可否チェック - Qiita". The snippet shows a Ruby code snippet: `2019/05/18 — Copied! class Child < ActiveRecord::Base belongs_to :parent # 追加 before_destroy :must_not_destroy_last_one_child private def must_not_destroy_last_one_child throw(:abort) if (parent.children - [self]).empty? end end ...`

The second result is from <https://railsguides.jp> with the title "Active Record コールバック - Railsガイド". The snippet shows: `3.3 オブジェクトのdestroy, before_destroy; around_destroy; after_destroy; after_commit/after_rollback, after_save コールバックは作成 ...`

Below the second result, there is a section titled "他の人はこちらも検索" (Others also search for) with a list of related search terms: "Before_destroy", "throw :abort", "rails コールバック 一覧", "Rails before_destroy prepend", "rails コールバック before_destroy admin", and "After_create_commit".

The third result is from <https://ryym.tokyo> with the title "Rails: before_destroyによる関連チェックに注意・ryym.log". The snippet shows: `2016/03/22 — そのために、before_destroyコールバックを登録する。ハマった事、before_destroy に登録したメソッドを単体で実行してみると正しく動くのに、いざdestroy ...`

The fourth result is from <https://gdgd-shinoyu.hatenablog.com> with the title "before_destroy時点でdestroyされてしまう、dependentの罠 ...". The snippet shows: `2015/04/23 — ... primary key # name :string(255) # class User < ActiveRecord::Base has_many :articles, dependent: :destroy after_create do articles.import(%w"1 2 3 4 5".map{|text| Article.new(text:text)}) end before_destroy do p arti...`

foremで検索してみる

cache と名の付くメソッドを実行していることが多いようです

メソッド名から察するにデータの削除と一緒にキャッシュに対して何かを行うようなことが多そうです

The screenshot shows the GitHub search interface for the query 'before_destroy'. The left sidebar displays repository statistics: Code (7), Commits (2), Issues (15), Discussions (Beta, 0), Packages (0), Wikis (0), and Languages (Ruby, 7). The main content area shows 7 code results. The first result is from `app/models/reaction.rb`, showing lines 36-38 with `before_destroy` calls. The second result is from `app/models/github_repo.rb`, showing lines 12-13 with `before_destroy` and `after_save` calls. The third result is from `app/models/user_block.rb`, showing lines 13-14 with `before_destroy` and `after_create` calls. The fourth result is from `app/models/follow.rb`, showing lines 31-33 with `before_destroy` and `after_create` calls. Each result snippet includes a Ruby icon, a 'Showing the top two matches' or 'Showing the top match' indicator, and a 'Last indexed' date.

before_destroy Pull requests Issues Marketplace Explore

Code 7
Commits 2
Issues 15
Discussions Beta 0
Packages 0
Wikis 0

Languages
Ruby 7

Advanced search Cheat sheet

7 code results in [forem/forem](#) or view [all results on GitHub](#)

[app/models/reaction.rb](#)

```
36   after_create :notify_slack_channel_about_vomit_reaction, if: -> { category == "vomit"
37   }
37   before_destroy :bust_reactable_cache_without_delay
38   before_destroy :update_reactable_without_delay, unless: :destroyed_by_association
```

● Ruby Showing the top two matches Last indexed on 21 May

[app/models/github_repo.rb](#)

```
10   scope :featured, -> { where(featured: true) }
11
12   before_destroy :clear_caches
13   after_save :clear_caches
14
15   # Update existing repository or create a new one with given params.
```

● Ruby Showing the top match Last indexed on 17 Apr

[app/models/user_block.rb](#)

```
11   counter_culture :blocked, column_name: "blocked_by_count"
12
13   after_create :bust_blocker_cache
14   before_destroy :bust_blocker_cache
```

● Ruby Showing the top match Last indexed on 17 Apr

[app/models/follow.rb](#)

```
30                                     column_names: COUNTER_CULTURE_COLUMNS_NAMES
31   before_save :calculate_points
32   after_create :send_email_notification
33   before_destroy :modify_chat_channel_status
```

● Ruby Showing the top match Last indexed on 17 Apr

Google検索と比べてみましたが**検索結果が全然違います**

もしGoogleでのみ検索していた場合は**削除可否に使うのか**と**思ってしまった**もおかしくないですね

ここで問題が.....

本来なら**複数の OSS から before_destory を検索したい**と思いますよね

ただ GitHub の検索機能だとそれぞれの OSS から検索しないといけないため**すごく非効率**です.....

そんなことでお困りのあなたに**私が行っている検索方法を共有**しようと思います！

VS Code のワークスペース機能を使用して検索する

そもそもワークスペースとは？

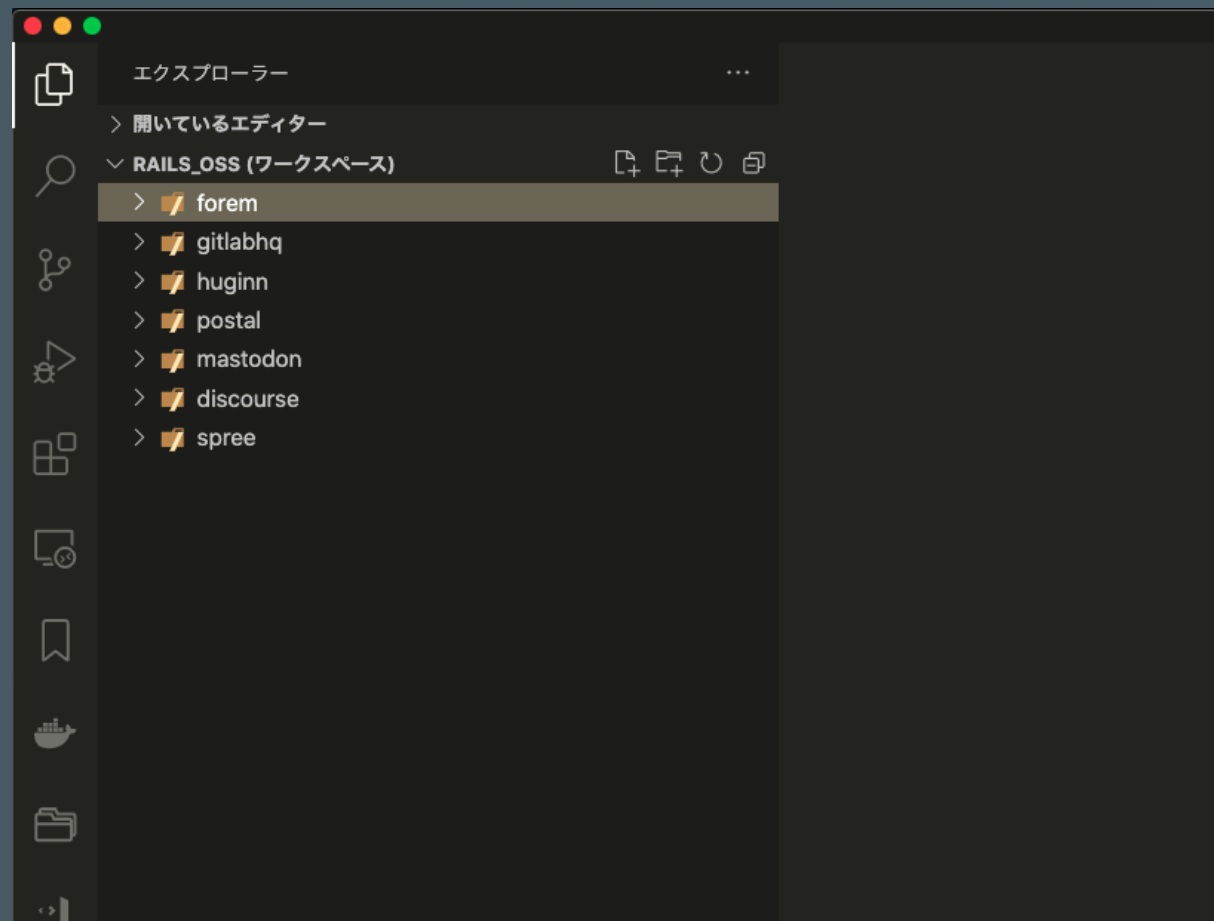
公式ドキュメントには以下のように書かれています

“ A Visual Studio Code "workspace" is the collection of one or more folders that are opened in a VS Code window (instance). In most cases, you will have a single folder opened as the workspace but, depending on your development workflow, you can include more than one folder, using an advanced configuration called Multi-root workspaces. ”

出典：[What is a VS Code "workspace"?](#)

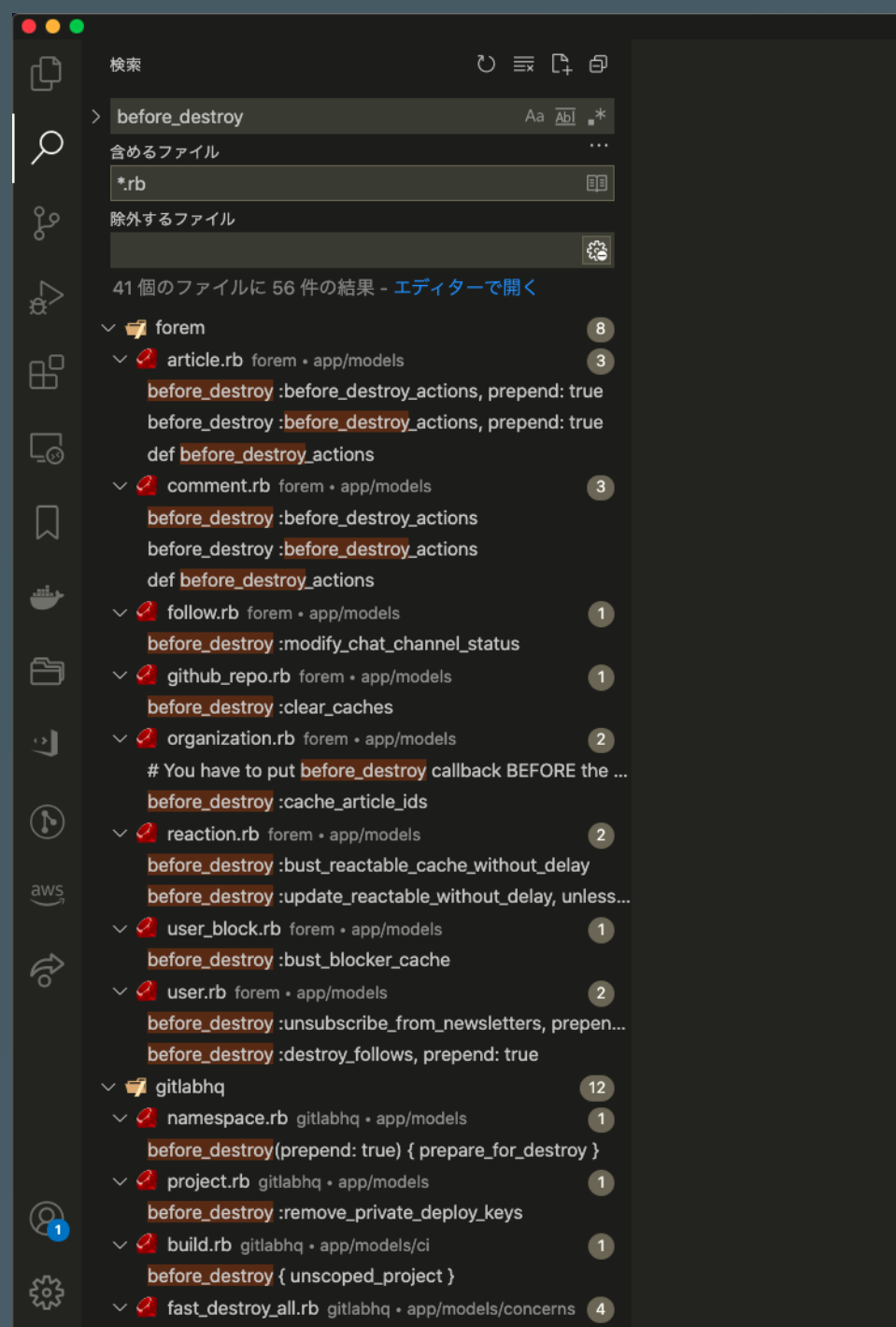
ウィンドウに**複数のルートフォルダを開けるようにする機能**です

複数の OSS をワークスペースに追加すると以下のような感じになります



実際に **before_destory** を
ワークスペースで検索して
みると以下のようになりま
す

複数の OSS から検索できる
ようになりました！ おめで
たうございます！



今日の発表を終わります



うん！？ ちょっと待って！



ローカルにそれぞれの OSS を git clone してきてそれをワークスペースに設定するのめんどくさくない？

OSS が更新したらどうすりゃいいんや.....

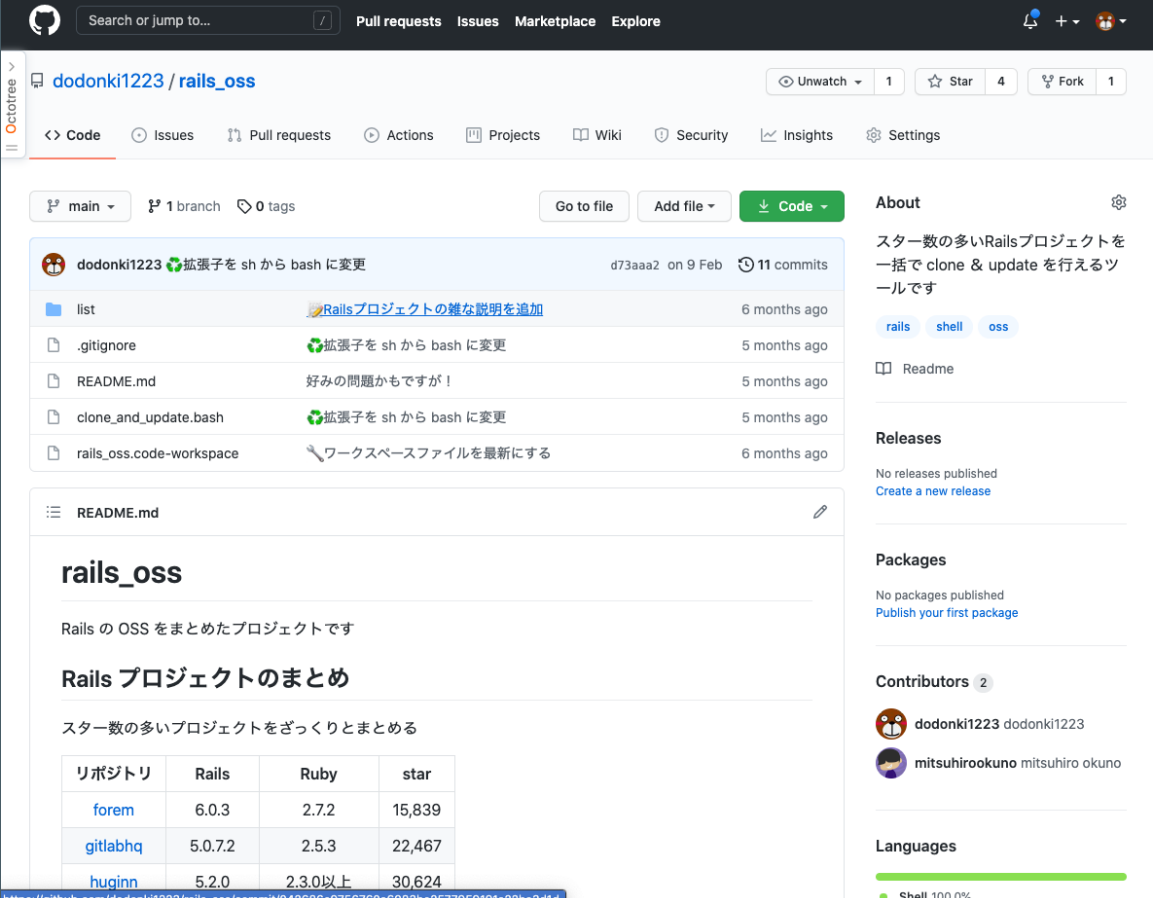
そんなことを思ったそのあなた！

**大丈夫です！ もっと簡単に取得できるように
しましょう！**

**私が以前、簡単にOSSを clone &
update できるツールを作成しました～**

Rails に関しては [rails_oss](#) リポジトリを clone してきて

bash clone_and_update.bash
を実行するだけで簡単にローカルに検索できる環境が出来上がります



The screenshot shows the GitHub repository page for `dodonki1223 / rails_oss`. The repository has 4 stars and 1 fork. The file list includes `list`, `.gitignore`, `README.md`, `clone_and_update.bash`, and `rails_oss.code-workspace`. The README content is as follows:

rails_oss

Rails の OSS をまとめたプロジェクトです

Rails プロジェクトのまとめ

スター数の多いプロジェクトをざっくりとまとめる

リポジトリ	Rails	Ruby	star
forem	6.0.3	2.7.2	15,839
gitlabhq	5.0.7.2	2.5.3	22,467
huginn	5.2.0	2.3.0以上	30,624

On the right sidebar, the "About" section states: "スター数の多いRailsプロジェクトを一括で clone & update を行えるツールです". The "Releases" and "Packages" sections indicate no releases or packages are published. The "Contributors" section lists `dodonki1223` and `mitsuhirookuno`. The "Languages" section shows "Shell" at 100.0%.

リポジトリの clone と update が終わった後は
oss.code-workspace ファイルを開き cmd +
shift + f で検索したい文字を打てば検索できる
ようになります

このツールは設定された OSS を clone & update し自動でワークスペースファイルを作成するツールになります

動作させると以下の
ような感じで自動で
clone と update が
行われるようになります

```
-bash
dodonki1223:~/project/rails_oss (main)
$ bash clone_and_update.bash
forem clone skip...
forem update
Your configuration specifies to merge with the ref 'refs/heads/master'
from the remote, but no such ref was fetched.
gitlabhq clone skip...
gitlabhq update
Updating e5f18314034..0c922a0a151
Updating files: 100% (7109/7109), done.
Fast-forward
 .eslintrc.yml                      9 +
 .gitignore                        1 +
 .gitlab/CODEOWNERS                10 +
 .gitlab/ci/build-images.gitlab-ci.yml 43 +
 .gitlab/ci/frontend.gitlab-ci.yml  68 +
 .gitlab/ci/global.gitlab-ci.yml    49 +
 .gitlab/ci/pages.gitlab-ci.yml     2 +
 .gitlab/ci/rails.gitlab-ci.yml     75 +
 .gitlab/ci/reports.gitlab-ci.yml   10 +
 .gitlab/ci/review.gitlab-ci.yml    70 +
 .gitlab/ci/rules.gitlab-ci.yml     92 +
 .gitlab/issue_templates/Design Sprint.md 1 +
 .gitlab/issue_templates/Feature Flag Removal.md 28 -
 .gitlab/issue_templates/Feature Flag Roll Out.md 17 +
 .gitlab/issue_templates/Geo Replicate a new Git repository type.md 116 +
 .gitlab/issue_templates/Geo Replicate a new blob type.md 118 +
 .gitlab/issue_templates/Security developer workflow.md 5 +
 .gitlab/issue_templates/Snowplow event tracking.md 2 +
 .gitlab/merge_request_templates/Documentation.md 69 +
 .gitlab/merge_request_templates/Pipeline Configuration.md 38 +
 .rubocop.yml                      6 +
 .rubocop_manual_todo.yml          412 +
 .rubocop_todo.yml                 7 +
 .stylelintrc                      1 +
 CHANGELOG.md                      746 +++
 GITALY_SERVER_VERSION              2 +
 GITLAB_KAS_VERSION                 2 +
 GITLAB_PAGES_VERSION               2 +
 GITLAB_SHELL_VERSION               2 +
 Gemfile                           41 +
 Gemfile.lock                       132 +
 LICENSE                            1 +
 README.md                          9 +
 VERSION                           2 +
 app/assets/images/aws-cloud-formation.png Bin 0 -> 2545 bytes
 app/assets/images/cluster_app_logos/fluentd.png Bin 2480 -> 0 bytes
 app/assets/images/mailers/members/issues.png Bin 0 -> 316 bytes
 app/assets/images/mailers/members/merge-request-open.png Bin 0 -> 327 bytes
 app/assets/images/mailers/members/users.png Bin 0 -> 371 bytes
 app/assets/javascripts/add_context_commits_modal/store/actions.js 10 +
 .../admin/application_settings/setup_metrics_and_profiling.js 3 +
```

**clone_and_update.bash が
何をやっているか説明していきます**

```
#!/bin/bash

declare -a cloneList=($(cat ./list/ssh))

echo '{
  "folders": [' > oss.code-workspace

for ((i = 0; i < ${#cloneList[@]}; i++)) {
  project_name=$(echo "${cloneList[i]}" | cut -d '/' -f2 | sed -e 's/.git//g')
  project_dir="./${project_name}"

  echo '
    {
      "path": "'${project_name}'"
    },' >> oss.code-workspace

  if [ ! -d $project_dir ]; then
    echo $project_name clone start...
    git clone ${cloneList[i]}
  else
    echo $project_name clone skip...
    echo $project_name update
    cd $project_name
    git fetch
    git pull
    cd ../
  fi
}

echo '  ],
  "settings": {}
}' >> oss.code-workspace
```

clone & update するリポジトリのリストを取得する

./list/ssh にリポジトリのリストが書かれており以下のコマンドでリストを取得します

```
declare -a cloneList=$(cat ./list/ssh))
```

./list/ssh には以下のようにリストを設定しています

```
git@github.com:forem/forem.git  
git@github.com:gitlabhq/gitlabhq.git  
git@github.com:huginn/huginn.git  
git@github.com:postalhq/postal.git  
git@github.com:tootsuite/mastodon.git  
git@github.com:discourse/discourse.git  
git@github.com:spree/spree.git
```

clone & update を行う

./list/ssh で取得したリスト分、繰り返し処理を行います
フォルダが存在しなければ clone を行い、フォルダが存在していれば update を行います

```
# ./list/ssh に追加したリポジトリ分繰り返す
for ((i = 0; i < ${#cloneList[@]}; i++)) {
    # git@github.com:oss_name/oss_name.git の形式からプロジェクト名を取得
    project_name=$(echo "${cloneList[i]}" | cut -d '/' -f2 | sed -e 's/.git//g')

    # プロジェクトのフォルダパスを取得
    project_dir="./${project_name}"

    # プロジェクトのディレクトリが存在するか
    if [ ! -d $project_dir ]; then
        # プロジェクトが存在しない場合は git clone を実行する
        echo $project_name clone start...
        git clone ${cloneList[i]}
    else
        # プロジェクトが存在している場合は git fetch & git pull を行う
        echo $project_name clone skip...
        echo $project_name update
        cd $project_name
        git fetch
        git pull
        cd ../
    fi
}
```

ワークスペースファイルを自動生成する

./list/ssh ファイルに記載されている git clone 用の URL からワークスペースのファイルを自動生成します

```
.  
.   
.   
echo '{  
    "folders": [' > oss.code-workspace  
.   
.   
.   
for ((i = 0; i < ${#cloneList[@]}; i++)) {  
.   
.   
.   
echo '        {  
            "path": "${project_name}"  
        }, ' >> oss.code-workspace  
.   
.   
.   
}  
  
echo '    ],  
    "settings": {}  
' >> oss.code-workspace
```

新しく OSS を追加する

新しく OSS を追加したい場合は ./list/ssh ファイルに git clone の URL を追加 します

./list/ssh ファイルに追加するだけでワークスペースファイルも更新されるので特に気にする必要はありません

素敵なOSSライフを！

今日の発表を終わります