

神戸大学 ITスペシャリスト養成コース
GitHubフロー実践

玉田春昭

京都産業大学

 @tama5

 tamada

 tamada@cc.kyoto-su.ac.jp

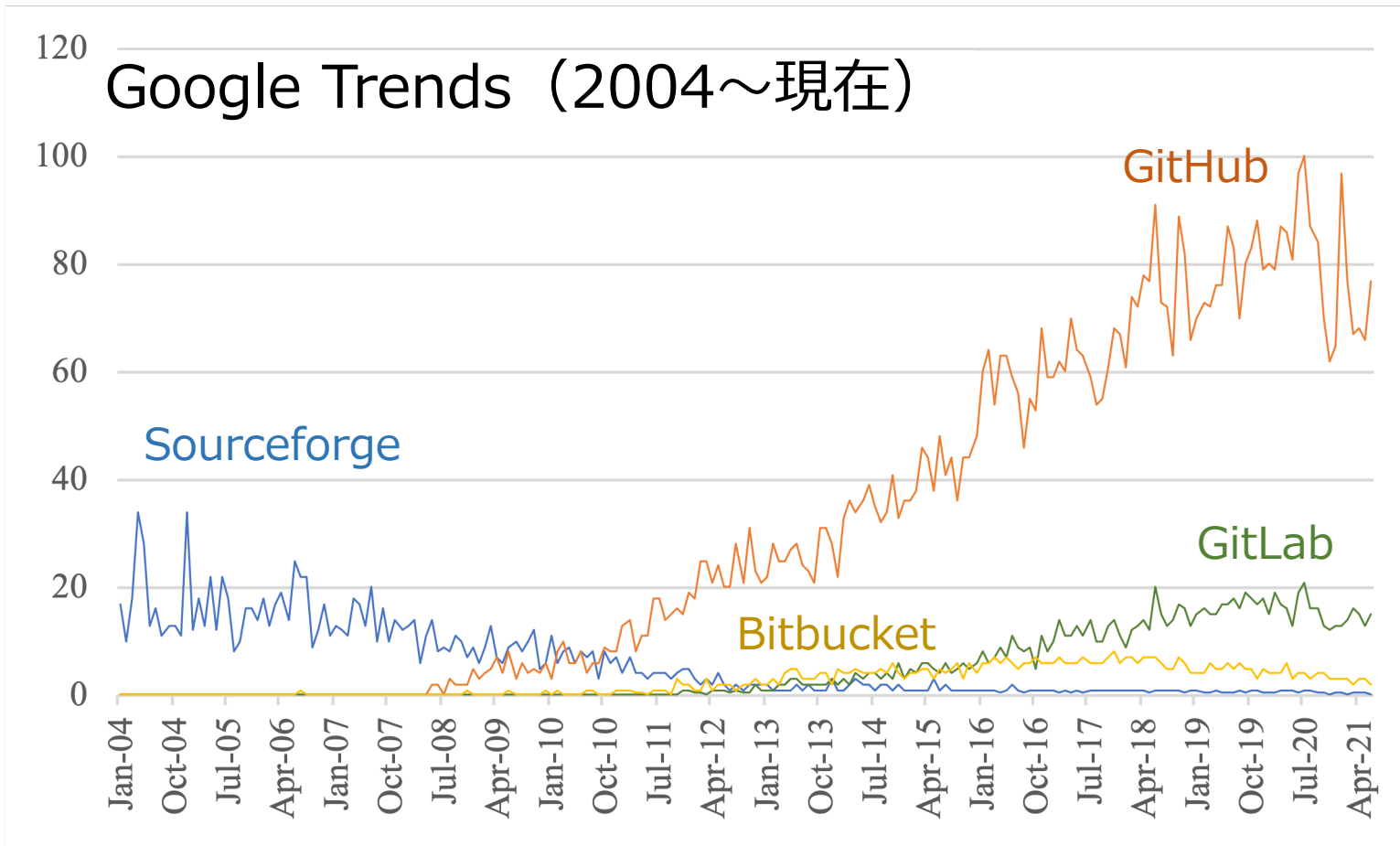
 <https://tamada.github.io>

本日の予定

1. GitHub flowの復習
2. 本日のお題の説明
3. GitHub flowの実践演習
4. まとめ

GitHub flowの復習

Social Codingが一般化



- 2008からGitHubがうなぎのぼり。
- Sourceforgeは2005から単調減少。

ブランチ戦略[1, 2]

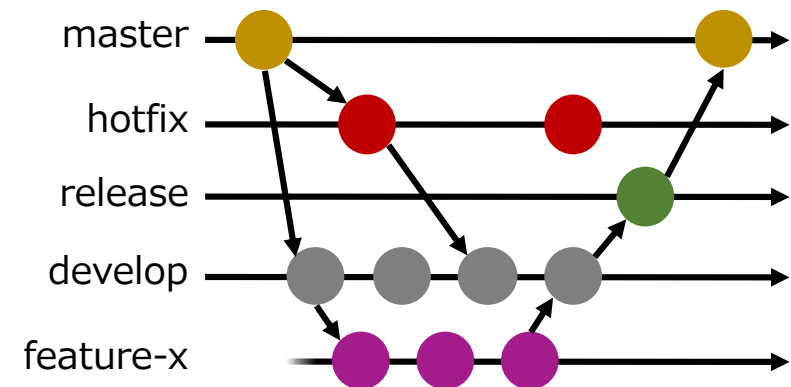
- ブランチ戦略の実施にはオーバーヘッドがある。
 - 一方で、ソフトウェアの品質に影響を与える。
 - ブランチ構造のミスはリリース後の故障率に影響する [2]。
- ブランチ戦略は必要だが、導入にはコストがかかる。

[1] Chuck Walrad, and Darrel Storm, “The importance of branching models in SCM,” Computer, Vol.35, Issue 9, pp.31—38, November 2002.

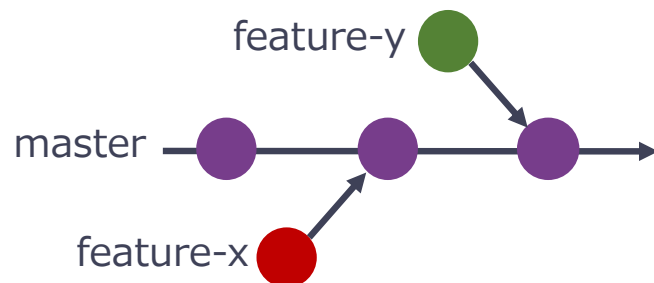
[2] Emad Shihab, Christian Bird, and Thomas Zimmermann, “The effect of branching strategies on software quality,” Proc. of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, September 2012.

代表的なブランチ戦略

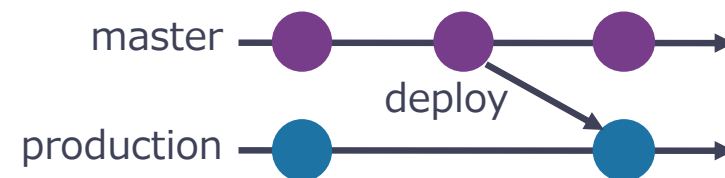
•Git flow



•GitHub flow



•GitLab flow



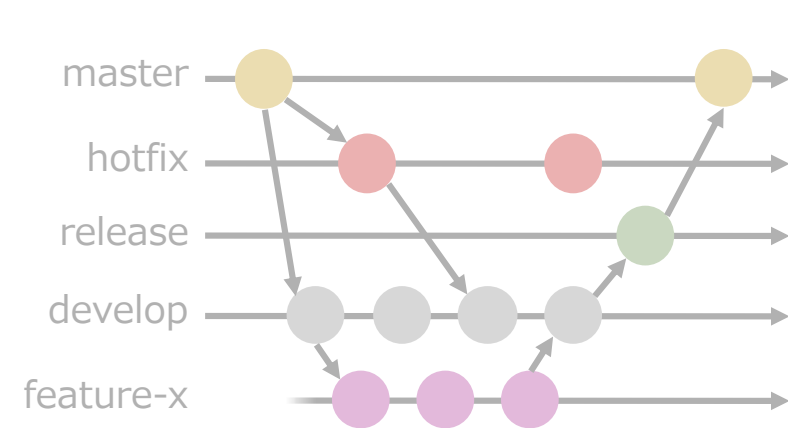
Git flow <https://nvie.com/posts/a-successful-git-branching-model/>

GitHub flow <https://scottchacon.com/2011/08/31/github-flow.html>

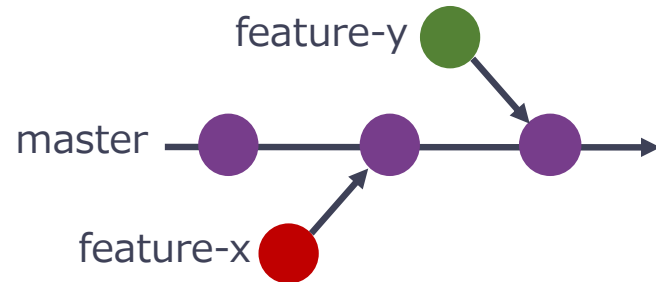
GitLab flow <https://about.gitlab.com/2014/09/29/gitlab-flow/>

本講義で利用するブランチ戦略

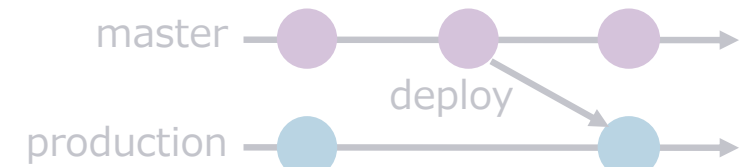
•Git flow



•GitHub flow



•GitLab flow

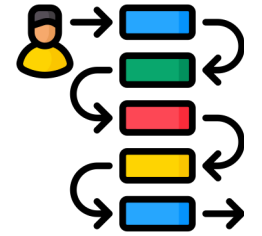


Git flow <https://nvie.com/posts/a-successful-git-branching-model/>

GitHub flow <https://scottchacon.com/2011/08/31/github-flow.html>

GitLab flow <https://about.gitlab.com/2014/09/29/gitlab-flow/>














GitHub flowのルール



1. mainブランチはいつでもデプロイ可能.
2. 作業用ブランチをmainブランチから作成する.
 - Ex. new-oauth, background_color, ...
3. 作業用ブランチを定期的にpushする.
4. プルリクエストを活用する.
5. プルリクエストが承認されればmainへマージする.
6. mainへのマージが完了したら直ちにデプロイする.

GitHub flowの手順

前提 プロジェクト
はClone済み

-   1. Issueを登録する.
-   2. ブランチを切る.
-  3. プログラムを修正する.
-   4. ローカル環境で動作を確認する.
-   5. プルリクエストを作成する.
-   6. マージする.
-   7. デプロイする.

GitHub flowの手順

前提 プロジェクト
はClone済み



1. Issueを登録する.



2. ブランチを切る.



3. プログラムを修正する.



4. ローカル環境で動作を確認する.



5. プルリクエストを作成する.



6. マージする.



7. デプロイする.

- これからどのような変更を行うのかを明確にするために、これからの作業のIssueを登録しよう.
- DoD (Definition of Done) を意識して Issueを書こう.
- 書き方に詰まれば周り
と相談しよう.

GitHub flowの手順

前提 プロジェクト
はClone済み



1. Issueを登録する.



2. ブランチを切る.



3. プログラムを修正する.



4. ローカル環境で動作を確認する.



5. プルリクエストを作成する.



6. マージする.



7. デプロイする.














- 次のコマンドでブランチを作成, 移動しよう.

```
git switch -c name
```

- ブランチを作成して移動する場合, -c オプションを付ける.
- ブランチがすでに存在している場合は, -c オプションは付けない.
- 現在のブランチを確認するときは, `git branch`

GitHub flowの手順














前提 プロジェクト
はClone済み

-   1. Issueを登録する.
-   2. ブランチを切る.
-  3. プログラムを修正する.
-   4. ローカル環境で動作を確認する.
-   5. プルリクエストを作成する.
-   6. マージする.
-   7. デプロイする.

- Visual Studio Codeを使い、プログラムを修正していこう.
- 先ほど登録したIssueのDoDを意識してプログラムを書いていこう.
- Issueにないことはグッと我慢しよう.

GitHub flowの手順








前提 プロジェクト
はClone済み

-   1. Issueを登録する.
-   2. ブランチを切る.
-  3. プログラムを修正する.
-   4. ローカル環境で動作を確認する.
-   5. プルリクエストを作成する.
-   6. マージする.
-   7. デプロイする.

- ローカル環境で動作確認をしよう.
- 単体テスト（自動テスト）があるとなお良い.
- 想定外の操作をしても大丈夫！？

GitHub flowの手順

前提 プロジェクト
はClone済み

-  1. Issueを登録する.
-  2. ブランチを切る.
-  3. プログラムを修正する.
-  4. ローカル環境で動作を確認する.
-  5. プルリクエストを作成する.
-  6. マージする.
-  7. デプロイする.

- 修正内容をGitHubにpushしてプルリクエストを作成しよう.
`git push origin branch_name`
- 正しく作成できているか、他の人に確認してもらおう.
 - 確認できたら、LGTM(Looks Good to Me)をもらおう.
 - 少なくとも2人に見てもらおう.

GitHub flowの手順

前提 プロジェクト
はClone済み



1. Issueを登録する.



2. ブランチを切る.



3. プログラムを修正する.



4. ローカル環境で動作を確認する.



5. プルリクエストを作成する.



6. マージする.



7. デプロイする.

- プルリクエストの画面でマージボタンを押してマージしよう.
- ローカル環境でmainブランチに移動し, GitHubからの変更を取り込もう.

```
git switch main  
git pull origin  
main
```

GitHub flowの手順

前提 プロジェクト
はClone済み



1. Issueを登録する.



2. ブランチを切る.



3. プログラムを修正する.



4. ローカル環境で動作を確認する.



5. プルリクエストを作成する.



6. マージする.



7. デプロイする.

- グループ全員がアクセスできる場所にデプロイしよう.
 - AWS
- 余裕があれば自動デプロイにも挑戦しよう.
 - GitHub Actions

本日のお題の説明

ToDoアプリを拡張する.

- 各人が少なくとも一つの拡張をGitHub flowで実施する.
 - 他の人のコードのレビューも実施する.
- 与えられるお題 (Issue) は解答例付き.
 - 与えられた以外のお題を実施することも可.

Issues

• 個人課題

解答例配布

1. 期限を入れられるようにする.
2. ToDoをキャンセルできるようにする.
3. ToDoの色（背景色）を変更可能にする.
4. ソート機能を導入する.
5. 日時を絶対表示から，相対表示に変更する.
6. ページ送り機能（Pagination）を追加する.
7. Docker対応にする.

この中から一人一つのIssueに取り組む。
誰がどのIssueに取り組むのかは，グループ内で相談して決めること。

• グループ課題

解答例なし

8. グループ独自の拡張を加える.
9. 自動デプロイを実現する.

個人課題が終了次第，グループ全員で取り組む。何をどの様に作り込むのかもメンバー間で相談する。

Issue 8. グループ独自の拡張を加える の例

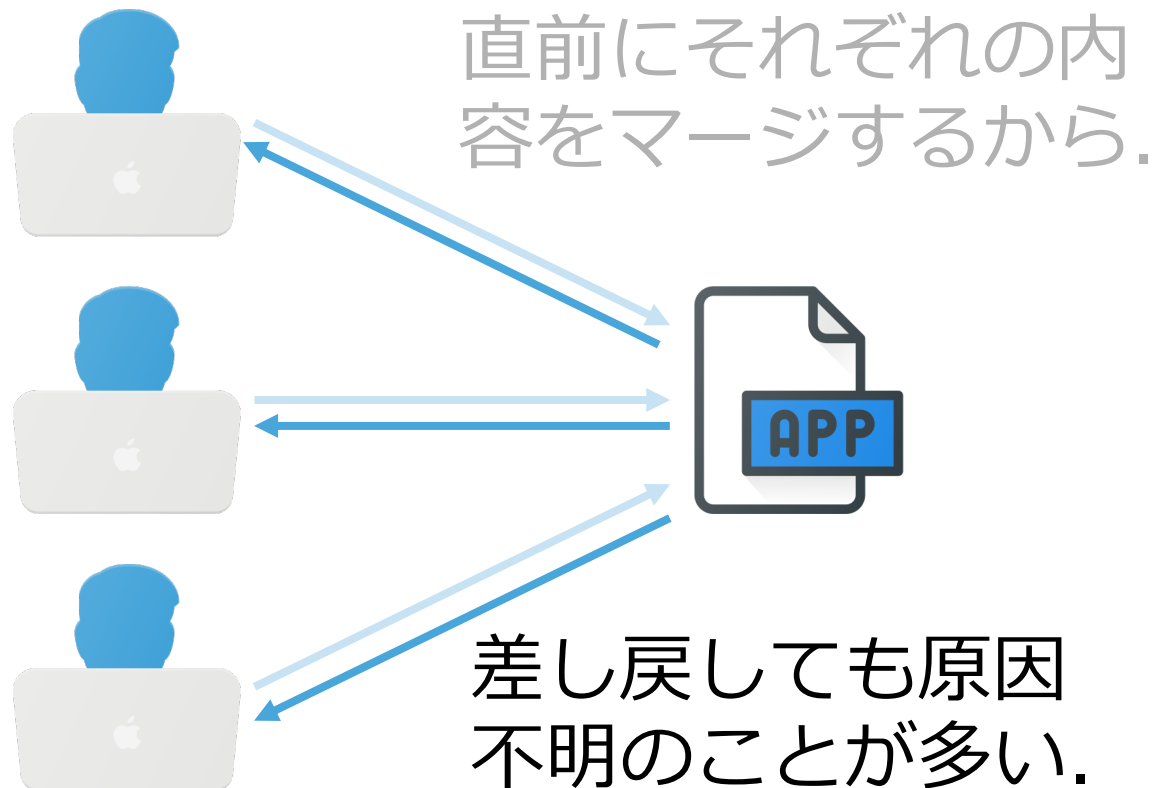
- UIデザインを変更する (CSS) .
- ページ送りの項目数をユーザ毎に設定可能にする.
- ToDoの文字色を変更可能にする.
- 期限を超えた/超えていないToDoのみを表示する.

Issue 9.自動デプロイを実現する.

- プルリクエストをマージしたら, mainブランチの内容をAWSに自動的にデプロイできるようにしよう.
 - GitHub Actionsを利用しよう.
 - <https://qiita.com/gdtypk/items/aea503db22dfc79f4e9e>
- **CI/CD**が実現できることになる.

これまでのチーム開発の困りごと

- 納期直前にビルドが壊れる。



- マージ時に不具合が出ることが多い。
 - 開発者間の差分が大きいと可能性が高くなる。
- マージはこまめに。
 - でも、手作業でのマージは辛い. . .

CI/CD (Continuous Integration/ Continuous Delivery)

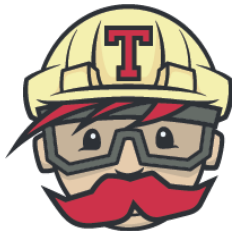
- クリーンビルド, テスト, デプロイを自動化する.
 - こまめにマージして, 開発者間の差分を小さくする.
 - 早期の不具合検出を可能にする.
 - 自動フィードバックが行われる安心感.
- 昨今の開発では必須とって良い.



GitHub Actions



Jenkins



Travis CI



DRONE
by harness

クリーンビルド

- 自分のPCとは異なる環境でビルドする。
 - PCの環境設定からビルドを実施する。
 - 自分の環境では動くよ, をなくす。
 - 異なるOSでの実行結果も確認できる。
 - 自分のPCは開発に専念できる。

CI/CDのアンチパターン

- 👎 CI/CDの設定を後回しにする。
- 👎 ローカル環境で長時間開発する。
- 👎 ビルドの失敗を放置する。

GitHub Actions



GitHub Actions

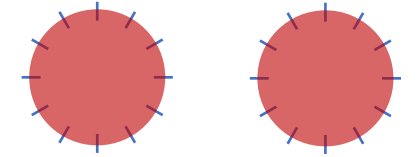
- GitHub標準のCI/CD.
- プロジェクトの`.github/workflows`ディレクトリ内に設定ファイルを置く.
 - 設定ファイルの文法などはWikiからのリンク先を参照のこと.

Issue 1～7 解答例に関する注意

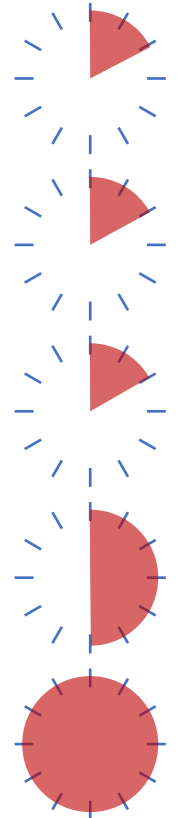
- 解答例を実際にマージする時、コンフリクトが起こる可能性がある。
 - 別のIssueで同じ行を追加していたり、同じ箇所に違う内容を書いていたりする。
 - 適切にマージすること。
 - 同じ内容が必要な場合は、コメントも同一のものになっている。
 - それ以外の場合は、両方の変更を適切に残すように変更すると良い。

GitHub flowの実践演習

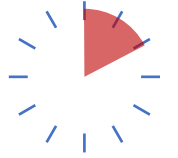
全体的な手順



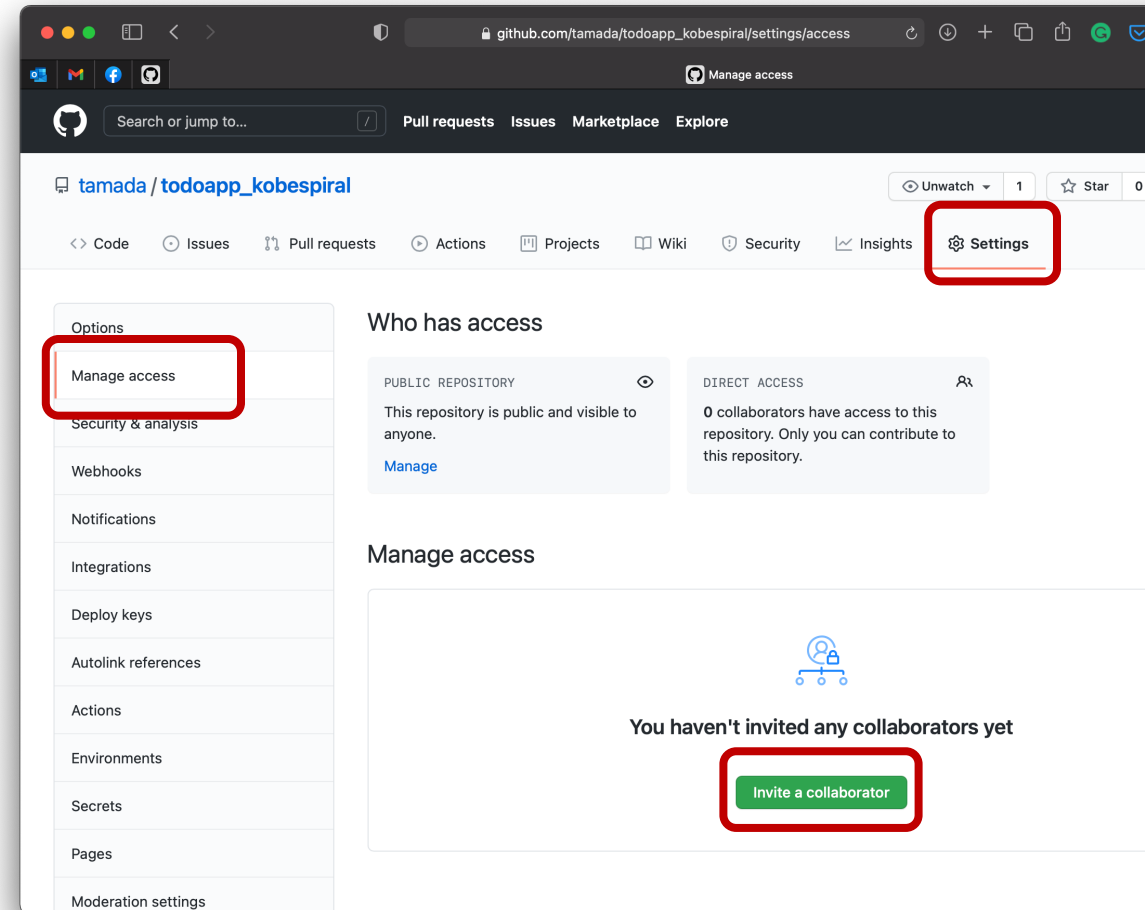
1. グループのリポジトリを作成する.
2. ソースコードを追加する.
3. 役割分担を決める (Issue 1~7) .
4. 各自Issueに取り組む.
5. Issue 8, 9に全員で取り組む.



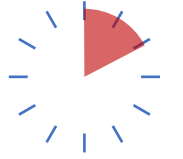
グループのリポジトリを作成する



- グループで共通のリポジトリを作成する.
 - グループの代表者がリポジトリを作成する.
 - todoapp_グループ名
 - リポジトリにグループメンバーを Collaborators として招待しよう.
- 作成できればソースコードを追加しよう.



ソースコードを追加する.

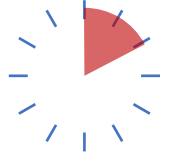


- 展開したディレクトリで次のコマンドを入力する.

```
git add .  
git commit -m "The first commit"  
git remote add origin git@github.com:...  
git push origin main
```

- Pushできれば, メンバ全員がローカルPCにクローンすること.

役割分担を決める.

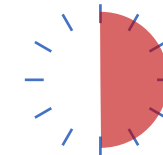


- Issue 1~7のうち, 誰がどのIssueを担当するのか決める.
 - レビュー担当者も決めよう.

Issues

- | 個人課題 解答例配布 | グループ課題 解答例なし |
|--|---|
| <ul style="list-style-type: none">1. 期限を入れられるようにする.2. ToDoをキャンセルできるようにする.3. ToDoの色 (背景色) を変更可能にする.4. ソート機能を導入する.5. 日時を絶対表示から, 相対表示に変更する.6. ページ送り機能 (Pagination) を追加する.7. Docker対応にする. <p>この中から一人一つのIssueに取り組む. 誰がどのIssueに取り組むのかは, グループ内で相談して決めること.</p> | <ul style="list-style-type: none">8. グループ独自の拡張を加える.9. 自動デプロイを実現する. <p>個人課題が終了次第, グループ全員で取り組む. 何をどの様に作り込むのかもメンバー間で相談する.</p> |








各自Issueに取り組む



- 解答例を参考に，Issueに取り組もう。
 - コンフリクトを適切に解決しよう。
 - レビューは紳士的に．絵文字も活用しよう。

GitHub flowの手順

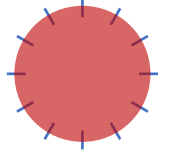
前提 プロジェクト
はClone済み

-  1. Issueを登録する。
-  2. ブランチを切る。
-  3. プログラムを修正する。
-  4. ローカル環境で動作を確認する。
-  5. プルリクエストを作成する。
-  6. マージする。
-  7. デプロイする。

Issue 1～7 解答例に関する注意

- 解答例を実際にマージする時，コンフリクトが起こる可能性がある。
 - 別のIssueで同じ行を追加していたり，同じ箇所に違う内容を書いていたりする。
 - 適切にマージすること。
 - 同じ内容が必要な場合は，コメントも同一のものになっている。
 - それ以外の場合は，両方の変更を適切に残すように変更すると良い。

Issue 8, 9に全員で 取り組む



- 方針を決めよう。
 - 完成形のイメージを全員で共有しよう。
- 誰がどんな作業を行うのかを決めよう。
- 作業に取り掛かろう。

心構え

- この演習の第一の目的は、GitHub flowの習得にある。
 - プログラムに注目するのではなく、プロセスに注目しよう。
 - グループメンバーとのGitHub上でのコラボレーションを楽しもう。

作業開始

Issues

個人課題

解答例配布

1. 期限を入れられるようにする。
2. ToDoをキャンセルできるようにする。
3. ToDoの色（背景色）を変更可能にする。
4. ソート機能を導入する。
5. 日時を絶対表示から、相対表示に変更する。
6. ページ送り機能（Pagination）を追加する。
7. Docker対応にする。

この中から一人一つのIssueに取り組む。誰がどのIssueに取り組むのかは、グループ内で相談して決めること。

グループ課題

解答例なし

8. グループ独自の拡張を加える。
9. 自動デプロイを実現する。

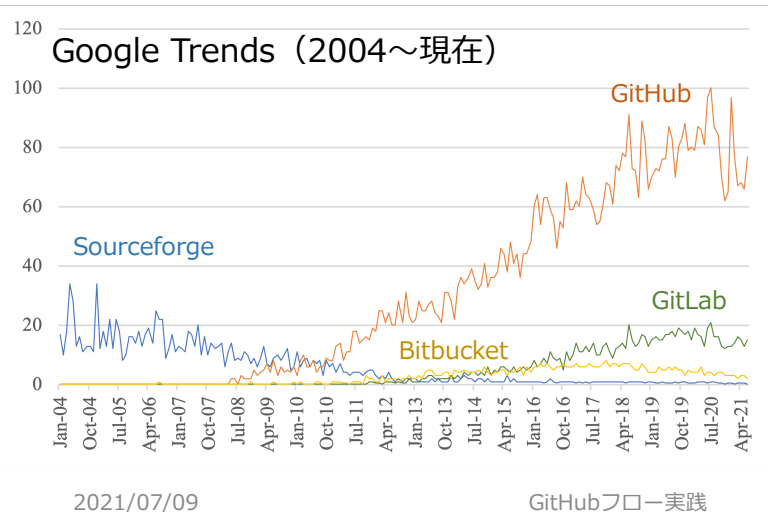
個人課題が終了次第、グループ全員で取り組む。何をどの様に作り込むのかもメンバー間で相談する。

全体的な手順

1. グループのリポジトリを作成する。
2. ソースコードを追加する。
3. 役割分担を決める（Issue 1～7）。
4. 各自Issueに取り組む。
5. Issue 8, 9に全員で取り組む。

まとめ

Social Codingが一般化



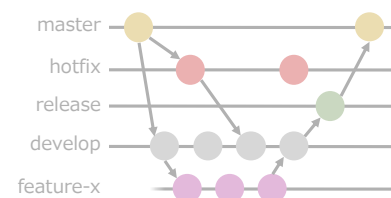
- 2008からGitHubがうなぎのぼり.
- Sourceforgeは2005から単調減少.

GitHubフロー実践

4

本講義で利用するブランチ戦略

•Git flow



Git flow

<https://nvie.com/posts/a-successful-git-branching-model/>

GitLab flow <https://scottchacon.com/2011/08/31/github-flow.html>

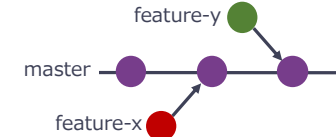
GitLab flow <https://about.gitlab.com/2014/09/29/gitlab-flow/>

2021/07/09

GitHubフロー実践

7

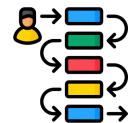
•GitHub flow



•GitLab flow



GitHub flowのルール



1. mainブランチはいつでもデプロイ可能.
2. 作業用ブランチをmainブランチから作成する.
 - Ex. new-oauth, background_color, ...
3. 作業用ブランチを定期的にpushする.
4. プルリクエストを活用する.
5. プルリクエストが承認されればmainへマージする.
6. mainへのマージが完了したら直ちにデプロイする.

2021/07/09

GitHubフロー実践

8

GitHub flowの手順

前提 プロジェクトはClone済み

1. Issueを登録する.
2. ブランチを切る.
3. プログラムを修正する.
4. ローカル環境で動作を確認する.
5. プルリクエストを作成する.
6. マージする.
7. デプロイする.

2021/07/09

GitHubフロー実践

9