セクション2: 準備

7. リモートリポジトリ作成

- リモートリポジトリはハイフンで繋ぐ

11. プロンプトをHackしよう

- `$ echo $SHELL`: 使用中のシェルを確認する
- `$ chsh -s /bin/bash`: シェルを変更する
- 以下、手順
  - `$ cp ~/.zshrc ~/.zshrc_bk`: バックアップを取る（あれば）
  - Oh My zshをインストール。コマンドはoh my zshのホームページから入手
  - $ sh -c "$(curl -fsSL https://raw.github.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
    - 参考: oh my zshをアンインストールする方法:
    - `$ rm -f ~/.oh-my-zsh`
    - .zshrc_bkを.zshrcに戻す
  - powerlevel10kをクローンする。powerlevel10kで検索して、GitHubのページからURLを入手
  - `$ git clone https://github.com/romkatv/powerlevel10k.git ~/.oh-my-zsh/custom/themes/powerlevel10k`
  - .zshrcを修正する: `subl ~/.zshrc`
    - `# ZSH_THEME="robbyrussell"` <--- コメントアウト
    - `ZSH_THEME="powerlevel10k/powerlevel10k"` <--- 追加

セクション3: Gitの基本ワークフロー

13. リポジトリをクローンする

- Gitユーザの設定
  - `$ git config --global user.name "comp8oe"`
  - `$ git config --global user.email "comp8oe@gmail.com"`
- Gitユーザの確認
  - `$ git config --global --list`
- リモートリポからcloneしてローカルリポを作成する
  - `$ git clone`
- ローカルリポに紐づいているリモートリポのURLを確認する
  - `$ git remote -v`
    - origin git@bitbucket.org:comp8oe/sample-repo-bitbucket.git (fetch)　プルする元？
    - origin git@bitbucket.org:comp8oe/sample-repo-bitbucket.git (push)　プッシュする先

14. 作業用のブランチを作成する

- ブランチ一覧を表示する

    - `$ git branch`
        - `* master`
        - *はチェックアウトしているブランチを指す
- 新しいブランチを切る
    - `$ git branch <branch_name>`
    - 単語はハイフンでつなぐ
- 作業するブランチを切り替える
    - `$ git checkout <branch-name>`
- 新しいブランチを作成して、チェックアウトする
    - `$ git checkout -b <branch-name>`
- pagerを無効にするやり方
    - `$ git config --global --replace-all core.pager "less -F -X"`

15. 作業内容をStageする

- 作業状態を確認する
    - `$ git status`
- README.mdを更新する。
    - $ git statusすると、以下のような表示になる。（抜粋）
        - `Change not staged for commit:`
        - `modified: README.md`
- 作業内容をステージに追加する
    - $ git add
    - $ git add . <--- 全てのファイル、フォルダをステージに追加する。
    - add後に$ git statusすると、表示が変わっている。
        - `Changes to be committed:`
        - `(use "git restore --staged <file>..." to unstage)`
        - `modified: README.md`

16. 作業内容をコミットする

- 作業内容をコミットする
    - `$ git commit -m "<commit message>"`

17. コミット履歴を見てみよう

- コミット履歴を見てみよう
    - `$ git log`
- GitHub/Bitbucketでリモートリポを作成したときにinitial commitが行われる
- originは、リモートリポジトリのこと。
- origin/mainは、リモートリポのmainブランチを指す。
- HEADはチェックアウトしているブランチのこと

19. pullしてpushする

- pushする前に一度リモートリポの内容を反映する(pullする)
    - `$ git pull <remote_ref> <branchname>`
    - <remote_ref>はgit remote -vで確認

        - origin git@**:/***.git (fetch)
        - origin git@**:/***.git (push)
        - ※ ブランチ名に注意！master or main
- リモートリポにローカルリポの内容を反映する(push)
    - `$ git push <remote_ref> <branchname>`

20. プルリクエストを作成してマージする

- 概要
    - pull requestを作成する。
    - pull requestを確認する。
    - pull requestをマージする。
    - mainブランチにてREADME.mdが更新されていることを確認する。
- リモートリポでマージする場合は、pull requeatというリクエストを出す
    - リモートリポはチームで共有するリポなので、自由にマージすることはできない。
    - pull requestは、merge requestだと思えばOK
    - 原則他のメンバーが、自分が出したpull requestを確認してマージする
    - pull requestは”PR"や”プルリク”と呼ぶことが多い
- GitHubの操作
    - pull requestsタブをクリック
    - New Pull Requestsをクリック
    - baseのブランチと、compare(比べる)のブランチをそれぞれ選択する。
    - ファイルの差分が表示される。
    - 問題なければ、Create Pull Requestをクリック
    - プルリクのタイトルとコメントを記入し、Create pull requestをクリック
    - ---以下マージ---
    - Pull requestsタブをクリック
    - リクエストをクリックする。
    - 内容をレビュー（確認）
    - レビューの仕方は別セクションで説明。基本はFiles Changedで差分を見る。下の方で、差分が確認できる。
    - 問題が無ければ、Conversationタブをクリックし Merge pull requestをクリックし、Confirm mergeボタンをクリックする
- Bitbucketの操作
    - リポジトリを選択
    - 左のメニューからブランチを選択
    - featureブランチを選択
    - 「プルリクエストを作成」ボタンをクリック
    - baseになるブランチと、compareのブランチを選択。　※GitHubとは左右が逆な感じ
    - Titleと説明を記入
    - プルリクエストを作成ボタンをクリック
    - ---マージ---

    - プルリクエストをクリック
    - 対象のリクエストをクリック
    - マージボタンをクリック
    - マージ

21. pull してローカルリポを更新する

- ローカルリポにて、mainブランチに移動する
    - `$ git checkout master`
- リモートリポのmainブランチの情報をローカルリポのmainブランチにpull
    - `$ git pull origin master`　- 注意: ローカルリポをマージするのではない!!!!!!!!!!!!
- ローカルリポのmainブランチのREADME.mdが更新されていることを確認する。
- ローカルリポのmainブランチは、リモートリポのmainブランチとできるだけ同期した方が良い。

22. 不要なブランチを削除する

- ローカルリポとリモートリポの両方から不要なブランチを削除する
- まずはローカルリポのupdate-readmeブランチを削除
    - `$ git branch -d <branch-name>`
    - mainにマージしていないブランチは削除できない
    - 強制削除する場合は-dではなく、-Dオプションを用いる
- リモートリポはブラウザ上で削除。GitHubの場合は、branchをクリックして、削除したいブランチのごみ箱アイコンをクリックする。

セクション4: Gitの基本操作を学習する

24. スクラッチでgitリポジトリを作成する

- スクラッチ = 最初から
- `$ git init <フォルダ名>`
- 例: `$ git init project-from-scratch`

25. 既存のディレクトリをgitリポジトリにする

- gitリポにしたいディレクトリに移動して、`git init`する
- 参考
    - $ rm -rf <フォルダ名>
    - -rは、フォルダの中のフォルダやファイルも一緒に削除する
    - -fは、強制削除

26. 既存ディレクトリをforkする

- 誰かのリモートリポジトリをコピーして、自分のリモートリポジトリにする＝fork
    - gitHubにログインする = https://github.com/D-S-Hub/git-practice にアクセス
    - forkボタンをクリック

    - 自分のリモートリポジトリのurlを取得する
    - `$ git clone <url>`

27. untrack fileを理解する

- track = 追跡
- Untracked files:　一度もaddしたり、commitしたことが無いファイル
- Changes not staged for commit:　既にトラックしているファイルを変更した場合
- gitがトラックしているファイルの一覧を表示
    - `$ git ls-files`

32. ファイル名の変更をGitで管理する(mv)

- `$ git mv <filename1> <filename2>`
- 通常のmvコマンドを使うと、削除して、新規追加したことになる。
- ファイルの削除をステージングエリアに追加するには$ git add -Aすればok
- git mvの方が楽。

33. ファイルの削除をGitで管理する(rm)

- `$ git rm <filename>`
- trackされていないファイルは削除できない
- ステージングエリアにあるファイルも削除できない。
- git rmすると削除したという作業がステージングエリアに残る。

34. コミット履歴を確認する(log)

- `$ git log`
- `$ git log --oneline`
- `$ git log --graph`
- `$ git log -- <filename>`
- `$ git log --follow`
- `$ git show <commitID>`

35. Gitの管理から外す(.gitignore)

- 外すファイルの例
    - サイズが大きいファイル
    - バイナリーファイル（画像、音声、動画など、テキストファイル以外）
    - 中間ファイル
    - パスワードなど外部に知られてはいけない情報を含むファイル
    - システムが生成するファイルやキャッシュファイル
- https://github.com/github/gitignore を参照しよう！！！
- よく使う書き方
    - 特定のファイル ---> output.log
    - ワイルドカード ---> *.csv
    - フォルダ ---> logs/

36. 今までの作業内容をリモートリポジトリにpushする

- `$ git pull origin main`
- `$ git push origin main`

セクション5: ブランチとマージ

38.ブランチの作成と名前変更と削除

- 特定の機能を開発したり、テスト用にブランチを切ることが多い。
- ブランチはコミットを指すただのポインタ
- `$ git branch -a`でリモートリポを含むローカルリポに存在する全てのブランチを表示する
- `$ git checkout <branchname>`ブランチの切り替え
- `$ git branch <branchname>`ブランチの作成
- `$ git branch -m <old> <new>` ブランチ名の変更
- `$ git branch -d` ブランチを削除
- ハイフンで区切る
- add-hogeなどが良い
- test, fix-bug, featureなどはよくない例

39.ブランチをマージする

- mainブランチのようにチームで共有するブランチにマージする場合は、リモートリポジトリでマージする。Pull Requestを出す。
- それ以外の場合、自分で切ったブランチにさらにブランチをきり、それをマージする場合。
- `$ git merge <branchname>`
- マージする前に差分を確認するとよい
- `$ git diff <base> <compare>`

注意点

`git merge`するときなbaseとするマージに、checkoutしてから行う！！！

40.automatic mergeをする

- Fast forwardマージ
- Fast forwardではないケースとは、マージ先に別のコミットがあるケース
- 各コミットが同じ部分を変更しているとうまくマージできない。Conflictが起きる。
- 各コミットが別の部分を変更していればマージできる。automatic mergeする。
- automatic mergeを体験してみよう。
- `$ git log --all --oneline`で、他のブランチのコミットも表示される。
- `$ git log --all --oneline --graph`で、少し視覚的になる。

45.リモートリポからローカルリポに情報をfetchする

- pull = fetch + merge
- `$ git fetch <remote_ref>`

- ローカルリポジトリの中にリモートリポジトリのブランチがある。
- 'git branch -a'で↑を確認できる
- ローカルリポ内のリモートリポの情報は古くなる。
- `$ git fetch` コマンドで↑の情報を更新できる。
- `$ git checkout remotes/origin/main`でローカルリポ内のリモートリポのmainに移動できる。

47.リモートリポからローカルリポにpullする

- `$ git pull <remote_ref> <branchname>`
- 指定したリモートリポ:remote_refの、指定したブランチ:branchnameを「今いるブランチ」にマージする
- 基本、pushするまえにpullする。ビハインドなブランチをpushすることはできない。エラーが起きる。

48.pull時のコンフリクトに対処する

- 業務ではmainブランチ上で作業を行わない。ブランチを切る。
- 定期的にリモートリポのmainブランチからpullすることで、後々mainブランチにマージしやすくなる。

1. github上でreadmeを変更
2. featureブランチでreadmeを変更
3. git pull origin mainするとコンフリクトが発生→指定したリモートリポ=origin, 指定したブランチ=main(リモートリポ上), マージ先=conflict_remote
4. readmeを修正し、git addし、git commitする

49.プルリクエストを作成する

- `git push origin conflict-remote`これによって、リモートリポジトリにconflict-remoteブランチが出来る。
- ブラウザで↑を確認。
- Compare & pull requestボタンをクリック
- デフォルトはフォークもとへのプルリクエストになっているので、注意
- commentを残す。レビューお願いします。
- create pull requestボタンをクリック
- 原則、リクエストに対するマージは他の人が行う。
- Files changedタブで変更内容を確認する。コードの横の+ボタンでコメントが書ける。
- review changesボタンをクリック.submit
- Conversationタブをクリックして、Merge pull requestボタンをクリック。必要に応じてコメントを書いて、confirm mergeボタンをクリック。
- ローカルリポのmainブランチにプルする。（やり方は割愛）

50.fork元のリポをローカルリポに登録する

- ローカルリポにfork元のリポ情報をセットする
  - `$ git remote add upstream <repo_url>`
- 以下、ブラウザの操作
  - 画面右上のD-S-Hub/git-practiceをクリック

    - Codeボタンからhttpsもしくはsshをクリップボードにコピー
- `$ git remote add upstream git@github.com:D-S-Hub/git-practice.git`

51.fork元のリポにPRを作成する

- 今回pullするが、pull先はfork元。
    - `$ git pull upstream main`
- ローカルリポからforkもとにプッシュしない。一旦自分のリモートリポへプッシュ。
    - `$ git push origin new-feature`
- fork元のリポジトリにプルリクエストを出す
    - Compare & pull requestボタンをクリック
    - base repositoryをfork元のリポジトリにする！
    - コメントはきちんと書く。

セクション7: 差分(diff)を使いこなす

53.セクション概要

- diff = 差分
- いろいろなdiffがある。working directoryとstaging areaとリポジトリ

54.リポジトリを準備する

- リモートリポジトリを新規作成する。

55.p4mergeをセットアップする

- p4merge = 無料で使えるdiffツール
- https://www.perforce.com/downloads/visual-merge-tool

セクション8: rebaseを使ってブランチを統合しよう

64. 基本的なrebaseのフロー

- Rebaseのメリット
    - 不要なマージコミットを作らない
    - コミット履歴がすっきりする
    - 将来マージするブランチを適宜rebaseしておくと、後々マージしやすくなる

セクション10: コミットにtag付けをしよう

74. シンプルな(lightweight)tagをつける

- `$ git tag <tagname>`: 最新のコミットにラベル付けをする
- `$ git tag --list`: tag一覧を表示する
- `$ git tag --delete <tagname>`: 指定したtagを削除する
- `$ git show <tagname>`: tagnameで指定したコミットの内容を表示する

75. アノテーション付きtag(annotated tag)をつける

- `$ git tag -a <tagname>`: annotated tagを作成する

76. tag同士のdiffを確認する

- `git diff <tagname1> <tagname2>`: tag同士のdiffを表示する

77. 特定のコミットにtag付けする

- `$ git tag -a <tagname> <commitID>`: 特定のコミットにtagを付ける
- 順番に注意。tagnameが先。

78. tag情報をリモートリポにpushする

- tagを付ける目的の一つは、バージョン情報の共有。リモートリポにtagを付けると効果を発揮。
- `$ git push <remote_ref> <tagname>`: 指定のタグをリモートリポに送信する
- 注意！！普通にgit push origin mainのようにpushするだけだと、tag情報はpushされない。
- `$ git push <remote_ref> --tags`: 全てのタグ情報をpushする
- `$ git push <remote_ref> :<tagname>`: 特定のtag情報をリモートリポから削除する
- tagを付けるとリモートリポジトリからzip等でダウンロードできるようになる。

79. 特定のtagにcheckoutする

- `$ git checkout tags/<tagname>`: コードを特定のバージョンの状態にする
- `$ git fetch --tag --all`: 全てのtag情報をローカルに取得する

80. まとめ

セクション12: おまけ

87. git-flowとGitHub flow

- git-flow
  - 規模が大きくて複雑
  - 講師曰く「こんな複雑なフローをちゃんと守って開発しているチームは見たことがない」

- Github flow
    - Github社が実際に使っているワークフロー
    - git-flowを簡略化したもの
    - 本講座のシナリオも概ねGithub flowに沿っている。
    - 講師曰く「僕が働いているチームでもGitHub flowを使っている。多分現場で一番メジャー。」
    - mainブランチと、複数のfeatureブランチ
- GitHub flowのポイント
    - mainブランチは常にデプロイ可能な状態にする
    - featureブランチはmainブランチから作成する
    - featureブランチは定期的にpushする
    - Pull Requestを使用してマージする。ローカルリポでは基本マージしない。
    - mainにマージされたら直ちにデプロイする