

# CLI と Git と Github

---

# Contents

---

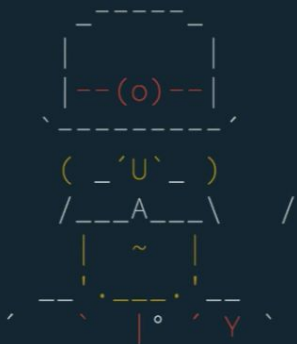
- CLIとコマンド操作
- Git
- Github

# CLIとコマンド操作

# CLIとGUI

# CLI(コマンドラインインターフェース)

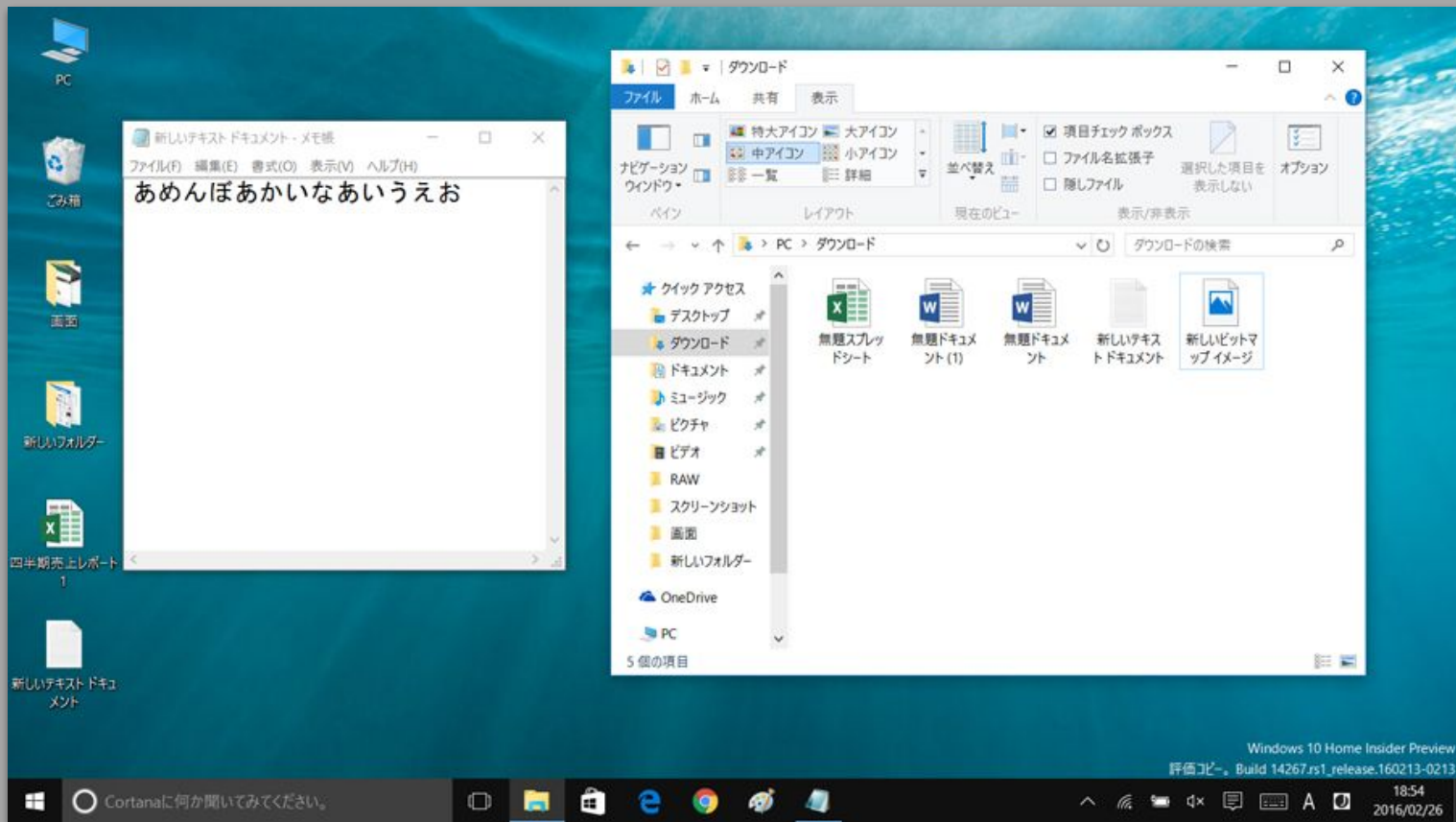
```
> npx oclif single kaomoji  
npx: installed 288 in 6.89s
```



Time to build a  
single-command CLI with  
oclif! Version: 1.3.10

```
? npm package name kaomoji  
? command bin name the CLI will export kaomoji  
? description Simple CLI to generate kaomoji!  
? author Chris Castle @crrcastle  
? version 0.0.0  
? license MIT  
? node version supported >=8.0.0  
? github owner of repository (https://github.com/OWNER/repo) crrcastle
```

# GUI(グラフィカルユーザーインターフェース)



# CLIとGUI

- もともとPCはCLIしかなかった.
- 現代のPCはGUI(の表示されたディスプレイ)を見ながら操作する.
- 一方で, サーバなどを操作する場合にはCUI(ターミナルなど)を使用する.
- 自分のPC内でも, 操作によってはCLIが必須な場合がある.



# サーバ

---

自分のPCのターミナルからサーバマシンにアクセスしてコマンド操作する様子





# 早速コマンドでPCを操作してみよう！

---

以下のアプリケーションを立ち上げよう！

これらのアプリケーションからコマンドを実行することができます！

- macの人 : ターミナル
- windowsの人 : Git bash

※以降、どちらも呼び方はターミナルで統一します！

# 【重要】作業フォルダという概念

---

ターミナルでコマンドを実行するときは必ず現在地（フォルダ）があります！

起動した直後の現在地を「ホームディレクトリ」と呼びます。

コマンドを実行するときは、適切なフォルダに移動する必要がある。

例：

- あるテキストファイルを編集するコマンドを実行したい場合、テキストファイルが入っているフォルダに移動しなければならない。

# 代表的な操作

---

```
// $マークが出ているときはコマンド待機中（入力できる状態）
```

```
// `cd`コマンド：最重要．作業ディレクトリを変更する．`cd ..`で1階層上に移動．
```

```
// `cd`まで入力して，フォルダのドラッグ&ドロップでも移動先を指定できる．
```

```
$ cd ディレクトリ名
```

```
// `ls`コマンド：現在のディレクトリにあるファイルとフォルダを表示する．
```

```
// -a をつけることで隠しファイルやフォルダを表示する．
```

```
$ ls
```

# ファイルやフォルダ作成

---

// `touch` コマンド : ファイルを作成する.

\$ touch 作りたいファイルの名前

// `mkdir` コマンド : フォルダを作成する.

\$ mkdir 作りたいフォルダの名前

// `cat` コマンド : ファイルの内容を表示する.

\$ cat 中身を表示したいファイルの名前

# CLI上のエディタとファイルの編集

---

```
// CLI上では`vi`と呼ばれるエディタを使用できる（他にもある）.  
// `vi`コマンドでエディタを起動する.
```

```
$ vi 編集したいファイルの名前
```

```
// エディタには「コマンドモード」と「インサートモード」がある.  
// 起動するとコマンドモードになる. ファイルの保存やエディタの終了などが行える.  
// `i`を入力するとインサートモードに変化する. `--INSERT--`などが表示される.  
// インサートモードの状態で, ファイルに対して文字列の入力や編集が行える.  
// `Esc`キーでコマンドモードに戻る.
```

# CLI上のエディタとファイルの編集

---

// コマンドモードで使用する代表的なコマンド

- :w 現在開いているファイルを保存.
- :q 現在開いているファイルを閉じる.
- :wq 上記2つの組み合わせ. 保存して閉じるの意味.
- :!q 保存せずに閉じる. 書いていて訳がわからなくなったらこれ.

# 練習

---

以下の処理をやってみよう！

CLI操作に慣れるのが目的！

- ターミナルでデスクトップに移動する.
- `test`フォルダを作成する.
- `test`フォルダに移動する.
- `test`フォルダ内に`sample.txt`ファイルを作成する.
- viで`sample.txt`を開き, `Hello CLI!`と記述して保存して閉じる.
- ターミナルで`sample.txt`の内容を表示する. (Hello CLIが表示されればOK)
- GUIで上記ファイルが作成されていることを確認.

# 【重要】滅びのコマンド`\$ sudo rm -rf /`

---

PCの全てが死ぬ. 絶対に実行してはならない.












(実行すると講義の受講に著しく支障をきたします)

参考: [https://news.mynavi.jp/article/dont\\_run\\_on\\_linux-11/](https://news.mynavi.jp/article/dont_run_on_linux-11/)



# Git

こんな経験はないかな??

-  20191012memo.txt
-  20191012memo1013再修正済.txt
-  20191012memo1013追記\_確認依頼.txt
-  20191012memo1013追記.txt
-  20191012memo修正.txt
-  20191012memo修正2\_チェック済.txt
-  20191012memo修正2.txt
-  20191012memo再修正.txt
-  20191012memo最終.txt
-  20191012memo追記.txt
-  memo.txt

これを防ぐのがGit

# Gitとはバージョン(変更履歴)を管理するツール

---

※ファイル自体は一つ！！

ver.01



ver.02



ver.03

コマンドで中身を記録  
ハッシュを発行

コマンドで中身を記録  
ハッシュを発行

コマンドで中身を記録  
ハッシュを発行

`git add .`  
`git commit -m"hoge"`

一連の履歴が全て記録される！！

# Gitとはバージョン(変更履歴)を管理するツール

前のバージョンに戻せる！

ver.01



ver.02



ver.03

コマンドで中身を記録  
ハッシュを発行

```
git add .  
git commit -m"hoge"
```

コマンドで中身を記録  
ハッシュを発行

ver.03で致命的なバグ

戻した履歴も全て記録される！！

これらは全て  
「自分のPCの中で」実行されます

「自分しか使えない」  
「PCが爆発したら終了」



# Github

Git ≠ Github

# Gitのバージョン管理をオンラインで共有できるのがGithub

リモート(web上)



ver.04までの変更履歴を  
オンラインにアップ!

git push



ローカル(自分のPC)

ver.01



ver.02



ver.03



ver.04

コマンドで中身を記録  
ハッシュを発行

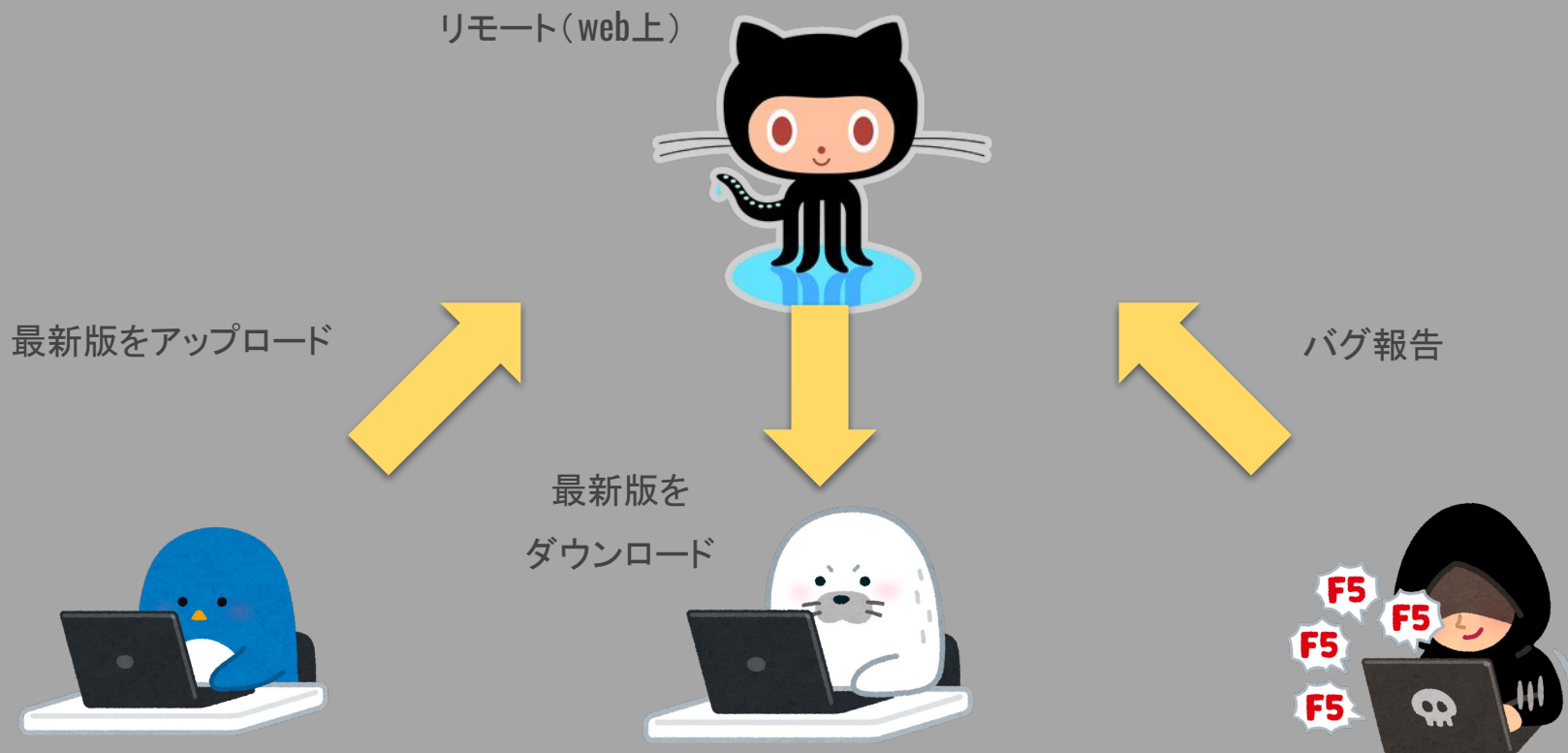
コマンドで中身を記録  
ハッシュを発行

コマンドで中身を記録  
ハッシュを発行

記録

# 複数人でのバージョン管理がより簡単に！

---



# Githubはエンジニア同士のSNS

- 開発者同士でのソースコード共有.
- 複数人でのプロダクト開発.
- 公開しているプロダクトの議論やバグ報告.

-> Githubは単なるバックアップツールではなく, 必須のSNS !!



# 有志による機能開発とプロダクトへの反映 (pull request)

Filters ▾

🔍 is:pr is:open

🔖 Labels 11

📅 Milestones 0

New pull request

🔗 57 Open ✓ 1,328 Closed

Author ▾

Label ▾

Projects ▾

Milestones ▾

Reviews ▾

Assignee ▾

Sort ▾

🔗 Bump @emotion/core from 10.0.35 to 10.1.0 in /packages/react-google-maps-api-storybook ✓ dependencies

#1643 opened 7 hours ago by dependabot-preview bot • Review required

🔗 Bump lint-staged from 10.3.0 to 10.5.1 ✓ dependencies

#1642 opened 7 hours ago by dependabot-preview bot • Review required

🔗 Bump eslint-plugin-react-perf from 3.2.4 to 3.3.0 in /packages/react-google-maps-api-storybook ✓ dependencies

#1641 opened 3 days ago by dependabot-preview bot • Review required

🔗 Bump webpack from 5.0.0 to 5.3.2 in /packages/react-google-maps-api-storybook ✓ dependencies

#1640 opened 3 days ago by dependabot-preview bot • Review required

🔗 Bump @storybook/react from 6.0.26 to 6.0.28 in /packages/react-google-maps-api-storybook ✓ dependencies

#1639 opened 3 days ago by dependabot-preview bot • Review required

🔗 Bump webpack from 5.0.0 to 5.3.2 in /packages/react-google-maps-api-marker-clusterer ✓ dependencies

#1638 opened 3 days ago by dependabot-preview bot • Review required

# バグ報告などコードを使ったコミュニケーション(issue)

```
8   ... "material-ui/system": "^4.9.10",
9 |   ... "react-google-maps/api": "1.8.7",
10  ... "testing-library/jest-dom": "4.2.4",
11  ... "testing-library/react": "9.5.0",
12  ... "testing-library/user-event": "7.2.1",
```

But it doesn't work. It was not updated at `/node_modules/@react-google-maps/api/src/components/streetview/StreetViewService.tsx`.  
(at line 39)

```
32  ... componentDidMount(): void {
33  ...  ... const streetViewService = new google.maps.StreetViewService()
34  ...
35  ...  ... this.setState(function setStreetViewService() {
36  ...  ...    ... return {
37  ...  ...      ... streetViewService,
38  ...  ...    }
39  ...  ... })
40  ... }
```



JustFly1984 commented on Apr 27 • edited

Owner



Tip



@taroosg released 1.8.9, please test again



1



1



1

# いままでに作成したプロダクトをGithubにアップしよう！

---

下記の資料にアクセス！

- [https://github.com/taroosg/github\\_tutorial](https://github.com/taroosg/github_tutorial)

上記資料の内容が完了したら次のページで`readme`ファイルを作成しよう！



# readmeファイルの作成

## 続いて、プロダクトの紹介ファイルを作ろう！

---

- エディタでプロダクトのフォルダを開き、`readme.md`を新規作成.
- 新規作成したreadmeファイルに次の内容を記述しよう！  
(readmeファイルはマークダウン形式で記述する)

# `readme.md`に記述(自分のプロダクトに合わせて記述)

---

# プロダクトのタイトル

## プロダクトの紹介

- 箇条書きにすると
- 読みやすい

## 工夫した点, こだわった点

- 特に見てほしい点,
- うまくできたと感じている点など.

## 苦戦した点, 共有したいハマリポイントなど

- ハマった経験は資産.
- あとで自分で見返しても有用.

# 記述が終わったら保存してGithubにpush

---

```
// ターミナルでコマンドを実行！  
// 【！！重要！！】必ず`cd`コマンドを実行  
  
$ cd プロダクトフォルダのパス  
$ git add .  
$ git commit -m"add readme file"  
$ git push origin master
```

## 【重要】ディレクトリ間違えた場合

---

コマンド`git init`を実行すると隠しフォルダ`.git`が作成される.

このフォルダの中に, バージョンの履歴などが保存されている.

誤ったフォルダで`git init`した場合は, `.git`フォルダを削除するとなかったことにできる.

→再度目的のディレクトリに移動して`git init`しよう！

# 【おまけ】デプロイ (HTML, CSS, JS限定)

# リポジトリのページから`Settings`にアクセス

---

<> Code    ! Issues    🔗 Pull requests    ▶ Actions    📁 Projects    📖 Wiki    🛡 Security    📈 Insights    ⚙ **Settings**

# 下の方にある`Github Pages`でSourceに`master`を設定

---


## GitHub Pages


GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

✓ Your site is published at <https://taroosg.github.io/win98janken/>

### Source

Your GitHub Pages site is currently being built from the `master` branch. [Learn more.](#)

 Branch: master ▼

 / (root) ▼

Save



URLが表示されるのでアクセス！

# 課題提出のまとめ

# 【まとめ】課題提出の流れ

---

1. 課題のプロダクトを各自実装する.
2. プロダクトのフォルダ内にreadmeファイルを作成し, 下記を記述する.
  - a. 概要(どんなものか, どうやって使うか, など)
  - b. エフした点, こたわった点
  - c. 苦戦した点, もう少し実装したかった点
  - d. 感想など(任意)
  - e. デプロイしていればURL
3. 作成したプロダクトをGithubにアップする.
4. 課題提出システムにURLを入力して送信 !

提出システム: <https://work-management-app.vercel.app/>