

# Agenda

## 【鈴木様 14回目レッスンの実施内容】

- ・ influxdb, コードレビュー
- ・ 次回までの宿題
  - ・ コーディング進める、dockerコマンド復習
- ・ 次回レッスン1/18(土) 21:30-

(2回目レッスン以降 基本的に 土曜日 21:30-)

## 前回の宿題

- ・ Docker インストール, コーディング進める

# Docker



## Docker コンテナ



Dockerは、**Linuxのコンテナ技術**を使ったもので、よく仮想マシンと比較されます。VirtualBoxなどの仮想マシンでは、ホストマシン上でハイパーバイザを利用しゲストOSを動かし、その上でミドルウェアなどを動かします。それに対し、コンテナはホストマシンのカーネルを利用し、プロセスやユーザなどを隔離することで、あたかも別のマシンが動いているかのように動かすことができます。そのため、**軽量で高速に起動、停止などが可能**です。

また、Dockerはミドルウェアのインストールや各種環境設定をコード化して管理します。

- 1.コード化されたファイルを共有することで、どこでも誰でも同じ環境が作れる。
- 2.作成した環境を配布しやすい。
- 3.スクラップ&ビルドが容易にできる。

# Docker

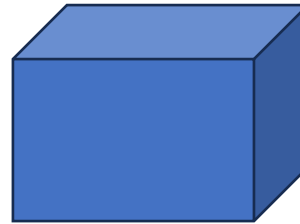
docker hub  
(リポジトリ)



pull



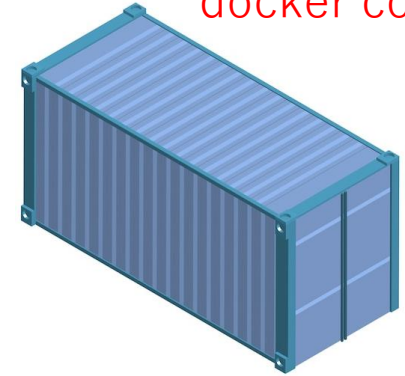
docker image



run



docker container

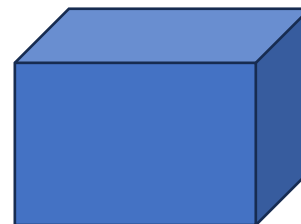


Dockerfile  
(カスタムイメージを  
作るためのレシピ)

build



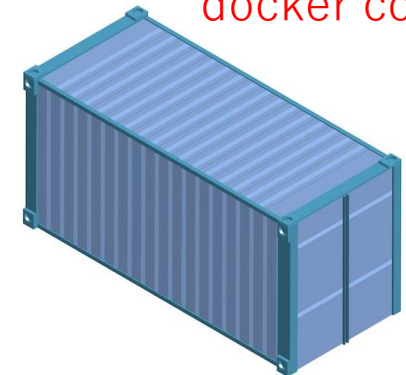
docker image  
(ユーザーカスタム)



run



docker container



# Docker

brew によるインストール

<https://blog.interstellar.co.jp/2022/03/14/installing-docker-on-a-mac-with-homebrew/>

pythonイメージダウンロード

```
docker pull python:3.11.7
```

django自作イメージビルド

```
docker build -t django .
```

django自作イメージからコンテナ起動

```
docker-compose up -d
```

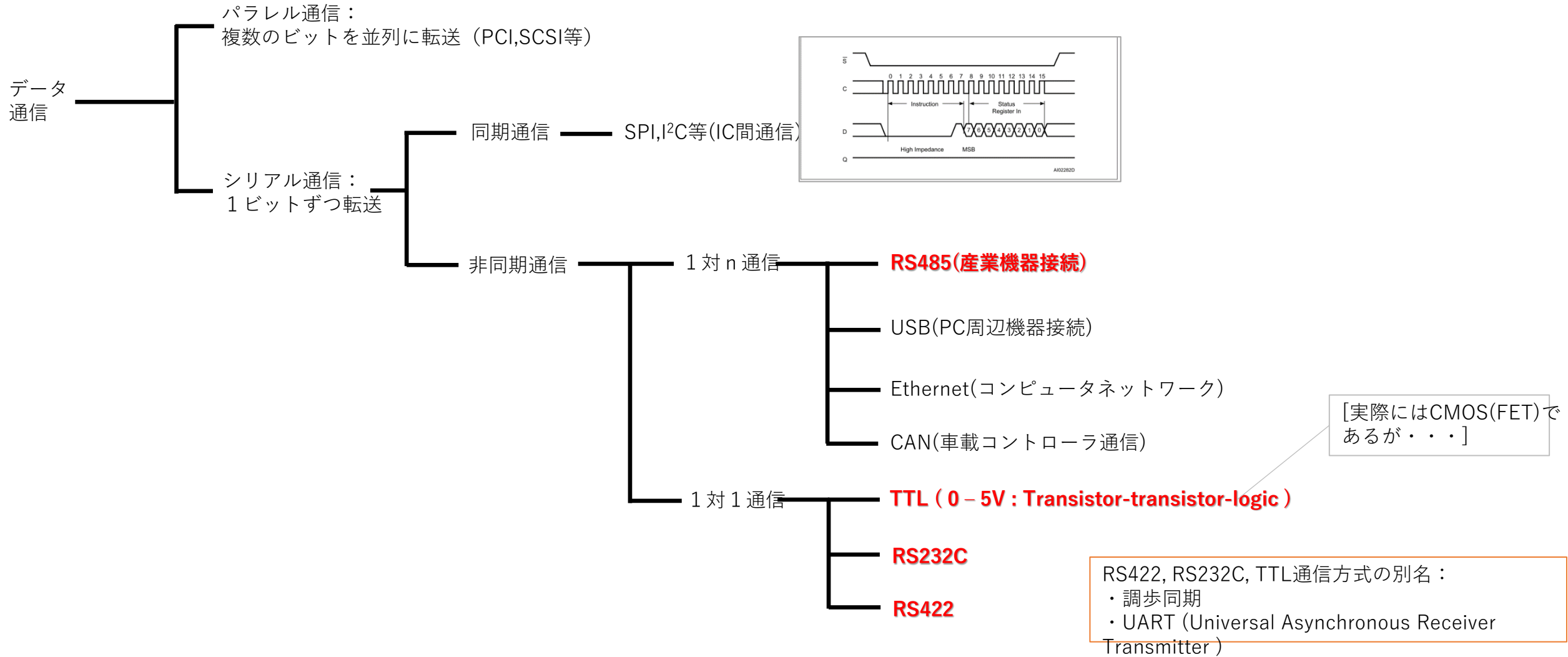
コンテナ停止

```
docker-compose down
```

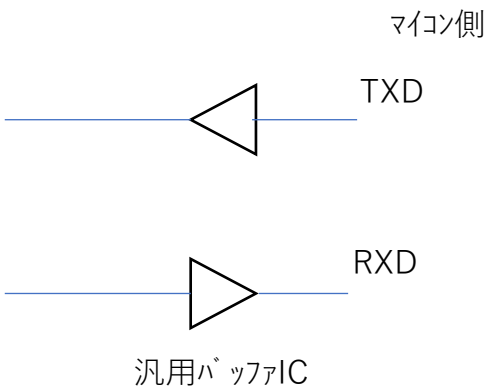
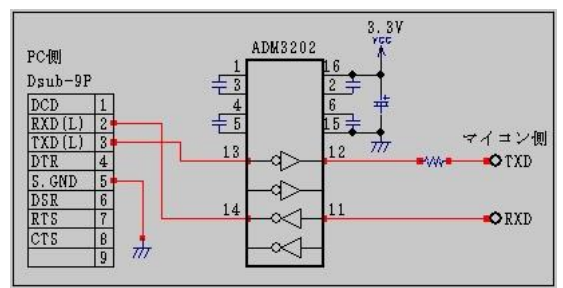
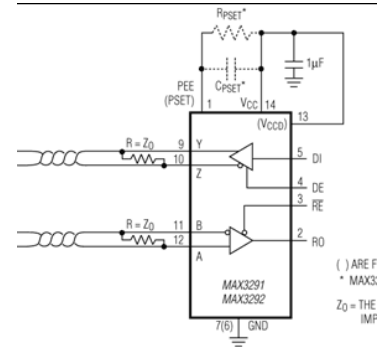
コンテナ内に入る場合

```
docker exec -it docker bash
```

## 代表的なデータ通信の種類と分類



# UART通信の電気的特性

|          | TTL(0-5V)   | RS232C  | RS422  |
|----------|---|---|--|
| 回路       | <p>マイコン側</p>  <p>汎用バッファIC</p> |  <p>専用IC必要</p> |  <p>専用IC必要<br/>(差動方式)</p> |
| H/L 認識電圧 | <p>H : +2.5V以上<br/>L : +1.5V以下</p>  | <p>H : +4 ~ +15V<br/>L : -4 ~ -15V</p>  | <p>+と-との差 (A/Y と B/Zの差)<br/>H : +0.2 ~ +5V<br/>L : -0.2 ~ -0.5V</p>  |
| 価格       | 安い  |   |  |
| 通信距離     | 短い  |   |  |
| 耐ノイズ性    | 低い  |   |  |

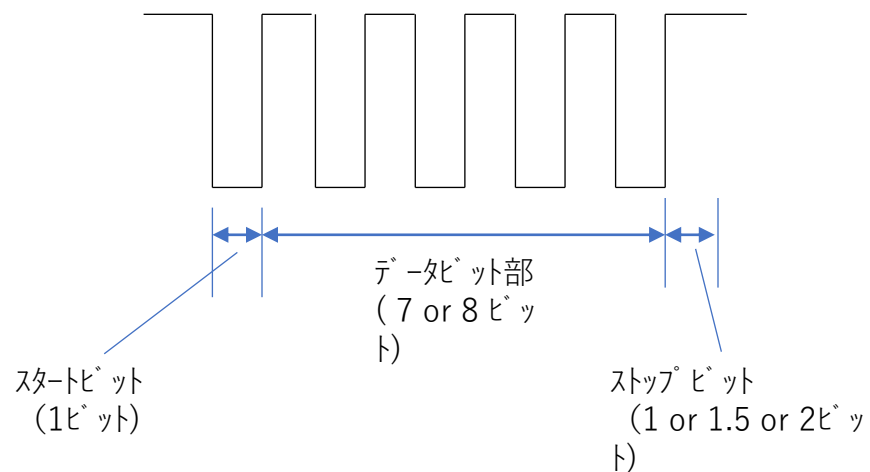
ここまでH か Lかを認識する電圧の違い  
(1ビットの認識)

## UART通信の1バイトデータの認識

通信データとしての最小単位：1バイト = 7 or 8 ビット

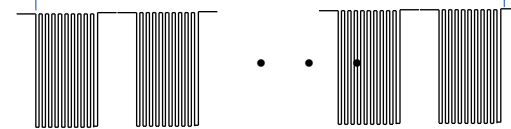
( $2^7-1$  : 0 ~ 127 /  $2^8-1$  : 0 ~ 255)

1バイト送信時の  
データビット構成



ボーレート : 単位 bps( bit  
per sec )

1秒間に何ビット送信するか



デフォルト通信ボーレート

ここまで1バイトを認識するための通信書式

## 1 バイトデータ：バイナリとASCIIの違い

- 例：文字 A を送る時実際には何を送信しているか

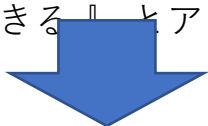


データビットが8ビットの時、0～255の数値として認識できる。



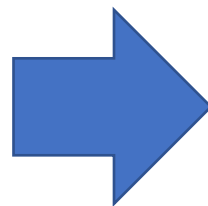
『文字Aを意味する数値を全世界で共通にすれば、その数値が』

文字であると認識できる』とアメリカ人が考えた



A～Z, a～z, 0～9(文字としての) 及び 制御文字(改行等)を

アメリカ人が標準化した



**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange  
(発音：アスキー)

### ASCII文字コード

| 文<br>字 | 10<br>進 | 16<br>進 | 文<br>字 | 10<br>進 | 16<br>進 | 文<br>字 | 10<br>進 | 16<br>進 | 文<br>字 | 10<br>進 | 16<br>進 | 文<br>字 | 10<br>進 | 16<br>進 | 文<br>字 | 10<br>進 | 16<br>進 | 文<br>字 | 10<br>進 | 16<br>進 |
|--------|---------|---------|--------|---------|---------|--------|---------|---------|--------|---------|---------|--------|---------|---------|--------|---------|---------|--------|---------|---------|
| NUL    | 0       | 00      | DLE    | 16      | 10      | SP     | 32      | 20      | @      | 64      | 40      | P      | 80      | 50      | `      | 96      | 60      | p      | 112     | 70      |
| SOH    | 1       | 01      | DC1    | 17      | 11      | !      | 33      | 21      | A      | 65      | 41      | Q      | 81      | 51      | a      | 97      | 61      | q      | 113     | 71      |
| STX    | 2       | 02      | DC2    | 18      | 12      | "      | 34      | 22      | B      | 66      | 42      | R      | 82      | 52      | b      | 98      | 62      | r      | 114     | 72      |
| ETX    | 3       | 03      | DC3    | 19      | 13      | #      | 35      | 23      | C      | 67      | 43      | S      | 83      | 53      | c      | 99      | 63      | s      | 115     | 73      |
| EOT    | 4       | 04      | DC4    | 20      | 14      | \$     | 36      | 24      | D      | 68      | 44      | T      | 84      | 54      | d      | 100     | 64      | t      | 116     | 74      |
| ENQ    | 5       | 05      | NAK    | 21      | 15      | %      | 37      | 25      | E      | 69      | 45      | U      | 85      | 55      | e      | 101     | 65      | u      | 117     | 75      |
| ACK    | 6       | 06      | SYN    | 22      | 16      | &      | 38      | 26      | F      | 70      | 46      | V      | 86      | 56      | f      | 102     | 66      | v      | 118     | 76      |
| BEL    | 7       | 07      | ETB    | 23      | 17      | '      | 39      | 27      | G      | 71      | 47      | W      | 87      | 57      | g      | 103     | 67      | w      | 119     | 77      |
| BS     | 8       | 08      | CAN    | 24      | 18      | (      | 40      | 28      | H      | 72      | 48      | X      | 88      | 58      | h      | 104     | 68      | x      | 120     | 78      |
| HT     | 9       | 09      | EM     | 25      | 19      | )      | 41      | 29      | I      | 73      | 49      | Y      | 89      | 59      | i      | 105     | 69      | y      | 121     | 79      |
| LF*    | 10      | 0a      | SUB    | 26      | 1a      | *      | 42      | 2a      | J      | 74      | 4a      | Z      | 90      | 5a      | j      | 106     | 6a      | z      | 122     | 7a      |
| VT     | 11      | 0b      | ESC    | 27      | 1b      | +      | 43      | 2b      | K      | 75      | 4b      | [      | 91      | 5b      | k      | 107     | 6b      | {      | 123     | 7b      |
| FF*    | 12      | 0c      | FS     | 28      | 1c      | ,      | 44      | 2c      | L      | 76      | 4c      | \      | 92      | 5c      | l      | 108     | 6c      |        | 124     | 7c      |
| CR     | 13      | 0d      | GS     | 29      | 1d      | -      | 45      | 2d      | M      | 77      | 4d      | ]      | 93      | 5d      | m      | 109     | 6d      | }      | 125     | 7d      |
| SO     | 14      | 0e      | RS     | 30      | 1e      | .      | 46      | 2e      | N      | 78      | 4e      | ^      | 94      | 5e      | n      | 110     | 6e      | ~      | 126     | 7e      |
| SI     | 15      | 0f      | US     | 31      | 1f      | /      | 47      | 2f      | O      | 79      | 4f      | _      | 95      | 5f      | o      | 111     | 6f      | DEL    | 127     | 7f      |

0(ゼロ) を数値0として送信する：バイナリ送信

0(ゼロ)を文字'0'(数値48)として送信する：ASCII送信



1 バイトデータ： バイナリとASCIIの違い（メリット  
デメリット）

**バイナリ送信のメリット** : 転送効率が良い  
(例：数値の100を送信するときは1バイトデータの100を送信  
すればよい)

**ASCII送信のメリット** : 数値と文字が混在して送信できる。  
制御文字（改行, STX(通信開始), ETX(通信終了) 等）が送信  
できる。

**ASCII送信のデメリット** : 転送効率が悪い  
(例：数値の100を送信したい場合は、文字 '6', '4')  
(10進数の100は16進数で0x64) の2バイト送信必要)

## まとめ

通信ができるためには

- ① 電気的特性( TTL / RS232C / RS422 )が一致している必要がある
- ② 通信書式 ( 1バイトのデータ構成: ボーレート, データビット数, ストップビット数, パリティ ) が一致している必要がある
- ③ 通信手順 ( データの始まり、終わりの認識方法 (STX,ETX)、データ整合性のチェック方法(BCC),その位置 ) が一致している必要がある
- ④ 通信コマンドの認識が一致している必要がある
- ⑤ 引数、データの意味合いの認識が一致している必要がある

OSI参照モデル

| 階層  | 名称             | 役割  |
|-----|----------------|---|
| 上位層 | 第7層 アプリケーション層  | ユーザーが直接操作するアプリケーション・ソフトに関する取り決め   |
|     | 第6層 プレゼンテーション層 | 通信のためのデータ形式とアプリケーション層でユーザーが取り扱うデータ形式(文字コード, 圧縮方式, 暗号化方式など)を相互に変換するための取り決め |
|     | 第5層 セッション層     | アプリケーションごとに、送信者と受信者が互いの存在を確認してからデータを送り合う(セッションの確立)するための取り決め               |
| 下位層 | 第4層 トランスポート層   | ネットワーク層以下の層で伝送されるデータが確実に受信者に届いていることを保証するための取り決め                           |
|     | 第3層 ネットワーク層    | 中継装置(ルーター)を経由して、データを最終的に目的地まで伝送するための取り決め                                  |
|     | 第2層 データリンク層    | 同じ種類の通信媒体(電線, 光ケーブル, 無線など)で直接つながっているコンピュータ同士でデータを伝送する際の取り決め               |
|     | 第1層 物理層        | 通信媒体に応じた信号の種類・内容やデータの伝送方法に関する取り決め   |

④⑤ → アプリケーション層, プレゼンテーション層

③ → セッション層

①② → データリンク層, 物理層

## 学習項目と順序

### ▼やりたいこと

0, ~~C言語の復習、C++~~/C#言語の学習

1, C# WINDOWSネイティブアプリケーションの開発

2, UART(RS232C, RS422等)でPCに接続した機器をWINDOWSネイティブアプリケーションから制御

3, 計測器ライブラリ: VISAの使い方

4, LANでPCに接続した計測器（電源、オシロスコープ、ロガー、電子負荷等）をWINDOWSネイティブアプリケーションから自動測定(VISAライブラリを使って)

5, それぞれ別々のインターフェースから測定したデータをデータベース(influxDB)に保存し、Grafanaで表示する

6, Git/Git labでのversion管理方法

C#を優先的に

外部ライブラリを使用しない  
シリアル通信から

## ▼やりたいこと

0, ~~C言語の復習、C++~~ **C#言語の学習**

**1, C# WINDOWSネイティブアプリケーションの開発**

2, UART(RS232C, RS422等)でPCに接続した機器をWINDOWSネイティブアプリケーションから制御

3, 計測器ライブラリ: VISAの使い方

4, LANでPCに接続した計測器（電源、オシロスコープ、ロガー、電子負荷等）をWINDOWSネイティブアプリケーションから自動測定(VISAライブラリを使って)

5, それぞれ別々のインターフェースから測定したデータをデータベース(influxDB)に保存し、Grafanaで表示する

6, Git/Git labでのversion管理方法

| 分類 |  |  |
|----|--|--|
| 入門 | ビルド環境, クラス基本, メソッド, 条件分岐, 繰り返し<br>コレクション, スコープ, 名前空間, 例外処理 |  |
| 基礎 | DLL分割, 継承, インターフェース, LINQ, ラムダ式,<br>非同期処理                  |  |

## 0. C#, 1. C# WINDOWSネイティブアプリケーションの開発

| 分類 |  | 備考 |
|----|--|----|
| 入門 | ビルド環境, クラス基本, メソッド, 条件分岐, 繰り返し<br>コレクション, スコープ, 名前空間, 例外処理 |    |
| 基礎 | DLL分割, 継承, インターフェース, LINQ, ラムダ式,<br>非同期処理                  |    |



[https://www.amazon.co.jp/dp/4798068330/ref=sspa\\_dk\\_detail\\_4?psc=1&pd\\_rd\\_i=4798068330&pd\\_rd\\_w=lmyoR&content-id=amzn1.sym.f293be60-50b7-49bc-95e8-931faf86ed1e&pf\\_rd\\_p=f293be60-50b7-49bc-95e8-931faf86ed1e&pf\\_rd\\_r=M4E58164FK419A1HWWX07&pd\\_rd\\_wg=Voomt&pd\\_rd\\_r=407ab2f4-2efc-429e-822b-54b79e44e096&s=books&sp\\_csd=d2lkZ2V0TmFtZT1zcF9kZXRhaWw](https://www.amazon.co.jp/dp/4798068330/ref=sspa_dk_detail_4?psc=1&pd_rd_i=4798068330&pd_rd_w=lmyoR&content-id=amzn1.sym.f293be60-50b7-49bc-95e8-931faf86ed1e&pf_rd_p=f293be60-50b7-49bc-95e8-931faf86ed1e&pf_rd_r=M4E58164FK419A1HWWX07&pd_rd_wg=Voomt&pd_rd_r=407ab2f4-2efc-429e-822b-54b79e44e096&s=books&sp_csd=d2lkZ2V0TmFtZT1zcF9kZXRhaWw)



[https://www.amazon.co.jp/%E7%8B%AC%E7%BF%92C-%E7%AC%AC5%E7%89%88-%E5%B1%B1%E7%94%B0-%E7%A5%A5%E5%AF%9B/dp/4798175560/ref=sr\\_1\\_1?\\_\\_mk\\_ja\\_JP=%E3%82%AB%E3%82%BF%E3%82%AB%E3%83%8A&crd=14SDVRZ44VTI1&dib=eyJ2IjojMSJ9.KLbcJQ5w4wKhILB0rzBvvtrszgIrFMhCQFwsUVfkqq2q27Cnl86VQLzdjSvJJUboTUNApn87RRsDeNs9hLegTrvLp1UlnRv-XkhawVatQcojKTuk4Bpt3nnFbKFP16gGhKqW22PNdCaho03szgHJI5GkSGrb8kCsPeoUjVJ2vGjE\\_8i5fmHH7M69V4TVztXH\\_VmZbv00KeiNdFtOL-tejti5X8VIVAS\\_YupsmX\\_Y8bSRdJ\\_80leDSFUHpwpaFTIfvvhidRHRuVuNhs5O0chSLwh07v8lYsch4pF65BYVKzKU.crq0fsCfNRZJPuzvPtyRHIO3S8-BZyqkvzI6UlyMgXs&dib\\_tag=se&keywords=C%23+%E7%8B%AC%E7%BF%92&qid=1726448554&srefix=c+%E7%8B%AC%E7%BF%92%2Caps%2C212&sr=8-1](https://www.amazon.co.jp/%E7%8B%AC%E7%BF%92C-%E7%AC%AC5%E7%89%88-%E5%B1%B1%E7%94%B0-%E7%A5%A5%E5%AF%9B/dp/4798175560/ref=sr_1_1?__mk_ja_JP=%E3%82%AB%E3%82%BF%E3%82%AB%E3%83%8A&crd=14SDVRZ44VTI1&dib=eyJ2IjojMSJ9.KLbcJQ5w4wKhILB0rzBvvtrszgIrFMhCQFwsUVfkqq2q27Cnl86VQLzdjSvJJUboTUNApn87RRsDeNs9hLegTrvLp1UlnRv-XkhawVatQcojKTuk4Bpt3nnFbKFP16gGhKqW22PNdCaho03szgHJI5GkSGrb8kCsPeoUjVJ2vGjE_8i5fmHH7M69V4TVztXH_VmZbv00KeiNdFtOL-tejti5X8VIVAS_YupsmX_Y8bSRdJ_80leDSFUHpwpaFTIfvvhidRHRuVuNhs5O0chSLwh07v8lYsch4pF65BYVKzKU.crq0fsCfNRZJPuzvPtyRHIO3S8-BZyqkvzI6UlyMgXs&dib_tag=se&keywords=C%23+%E7%8B%AC%E7%BF%92&qid=1726448554&srefix=c+%E7%8B%AC%E7%BF%92%2Caps%2C212&sr=8-1)

## 2. UART

### ▼やりたいこと

0, ~~C言語の復習~~、~~C++~~/**C#言語の学習**

1, C# WINDOWSネイティブアプリケーションの開発

2, UART(RS232C, RS422等)でPCに接続した機器をWINDOWSネイティブアプリケーションから制御

3, 計測器ライブラリ: VISAの使い方

4, LANでPCに接続した計測器（電源、オシロスコープ、ロガー、電子負荷等）をWINDOWSネイティブアプリケーションから自動測定(VISAライブラリを使って)

5, それぞれ別々のインターフェースから測定したデータをデータベース(influxDB)に保存し、Grafanaで表示する

6, Git/Git labでのversion管理方法



### 3. 4. VISA

#### ▼やりたいこと

0, ~~C言語の復習~~、~~C++~~/**C#言語の学習**

1, C# WINDOWSネイティブアプリケーションの開発

2, UART(RS232C, RS422等)でPCに接続した機器をWINDOWSネイティブアプリケーションから制御

3, 計測器ライブラリ: **VISA**の使い方

4, LANでPCに接続した計測器（電源、オシロスコープ、ロガー、電子負荷等）を**WINDOWS**ネイティブアプリケーションから自動測定(**VISA**ライブラリを使って)

5, それぞれ別々のインターフェースから測定したデータをデータベース(influxDB)に保存し、Grafanaで表示する

6, Git/Git labでのversion管理方法





### ▼やりたいこと

0, ~~C言語の復習~~、~~C++~~ **C#言語の学習**

1, C# WINDOWSネイティブアプリケーションの開発

2, UART(RS232C, RS422等)でPCに接続した機器をWINDOWSネイティブアプリケーションから制御

3, 計測器ライブラリ: VISAの使い方

4, LANでPCに接続した計測器（電源、オシロスコープ、ロガー、電子負荷等）をWINDOWSネイティブアプリケーションから自動測定(VISAライブラリを使って)

5, それぞれ別々のインターフェースから測定したデータをデータベース(influxDB)に保存し、Grafanaで表示する

6, Git/Git labでのversion管理方法



InfluxDB



Grafana



- Windowsローカルにdockerで構築
- InfluxDBには.NETドライバ有り



### ▼やりたいこと

0, ~~C言語の復習~~、~~C++~~/**C#言語の学習**

1, C# WINDOWSネイティブアプリケーションの開発

2, UART(RS232C, RS422等)でPCに接続した機器をWINDOWSネイティブアプリケーションから制御

3, 計測器ライブラリ: VISAの使い方

4, LANでPCに接続した計測器（電源、オシロスコープ、ロガー、電子負荷等）をWINDOWSネイティブアプリケーションから自動測定(VISAライブラリを使って)

5, それぞれ別々のインターフェースから測定したデータをデータベース(influxDB)に保存し、Grafanaで表示する

6, Git/Git labでのversion管理方法



- ・『gitによるソースコード管理』手法を学習したいか  
『ソフトウェア開発フロー』を学びたいかでGitLabである必要であるかが変わる。

前者であればGitLabは単なるリモートリポジトリなのでSAMURAI教材ベースのGitHubでよいのでは？

## 期間

### ▼やりたいこと

#### 0, ~~C言語の復習~~、~~C++~~、**C#言語の学習**

#### 1, C# WINDOWSネイティブアプリケーションの開発

#### 2, UART(RS232C, RS422等)でPCに接続した機器をWINDOWSネイティブアプリケーションから制御

#### 3, 計測器ライブラリ: VISAの使い方

#### 4, LANでPCに接続した計測器（電源、オシロスコープ、ロガー、電子負荷等）をWINDOWSネイティブアプリケーションから自動測定(VISAライブラリを使って)

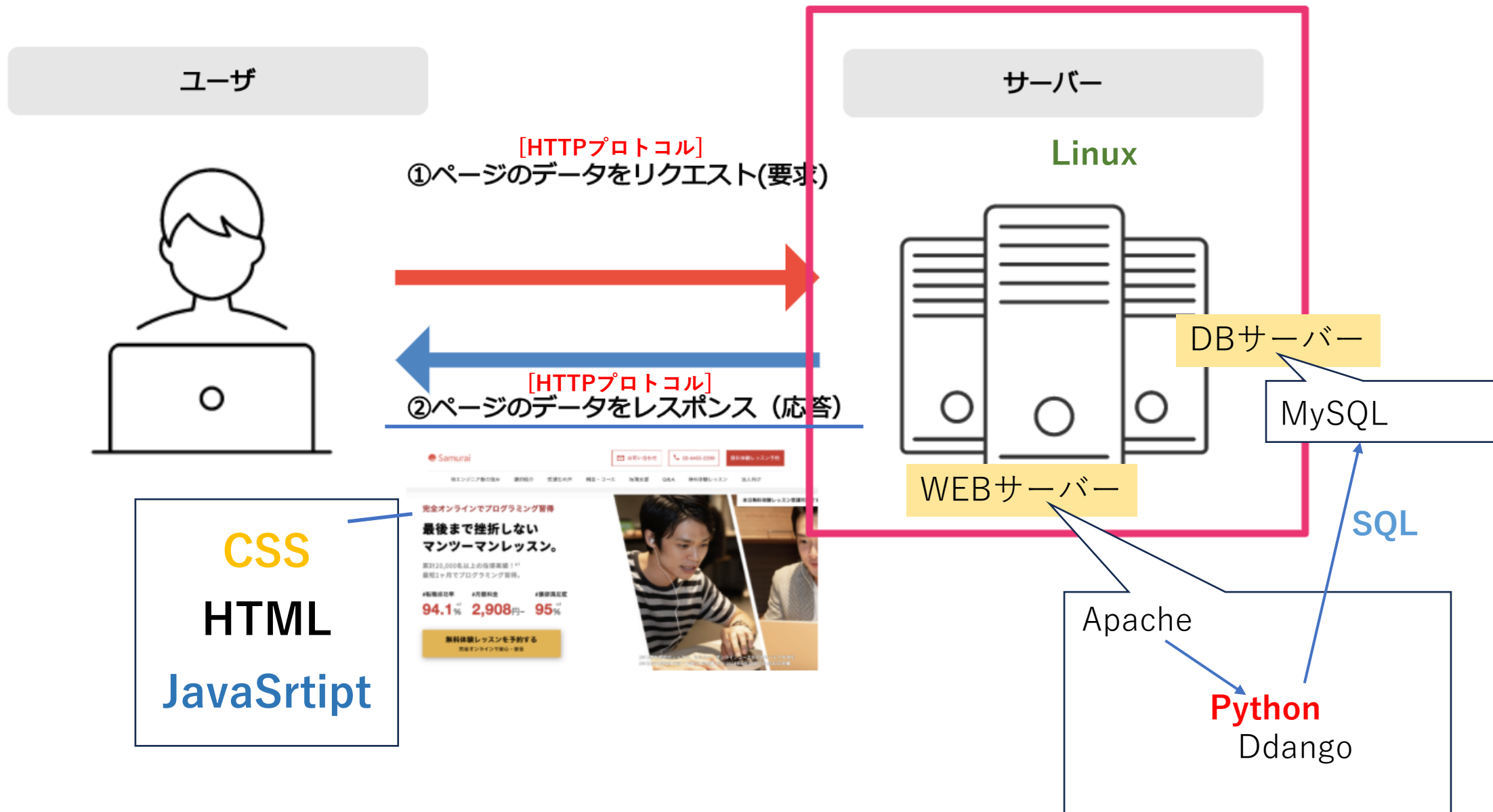
#### 5, それぞれ別々のインターフェースから測定したデータをデータベース(influxDB)に保存し、Grafanaで表示する

#### 6, Git/Git labでのversion管理方法

- ・ 必要な学習期間は現在の知識, ポテンシャル, どこまで深く学習するかで変わってくる
- ・ 1～2時間/日 => 約10時間/週 でプログラミング経験が少ない場合、一般論として3～4ヶ月で終わらせるのは困難。

| 項目                | 推定学習時間 |
|-------------------|--------|
| C#                | 100H   |
| UARTとそのアプリ        | 50H    |
| VISAとそのアプリ        | 50H    |
| influxDB, Grafana | 50H    |
| Git               | 20H    |

約270H :  $270 / 40 = 6.7$ ヶ月



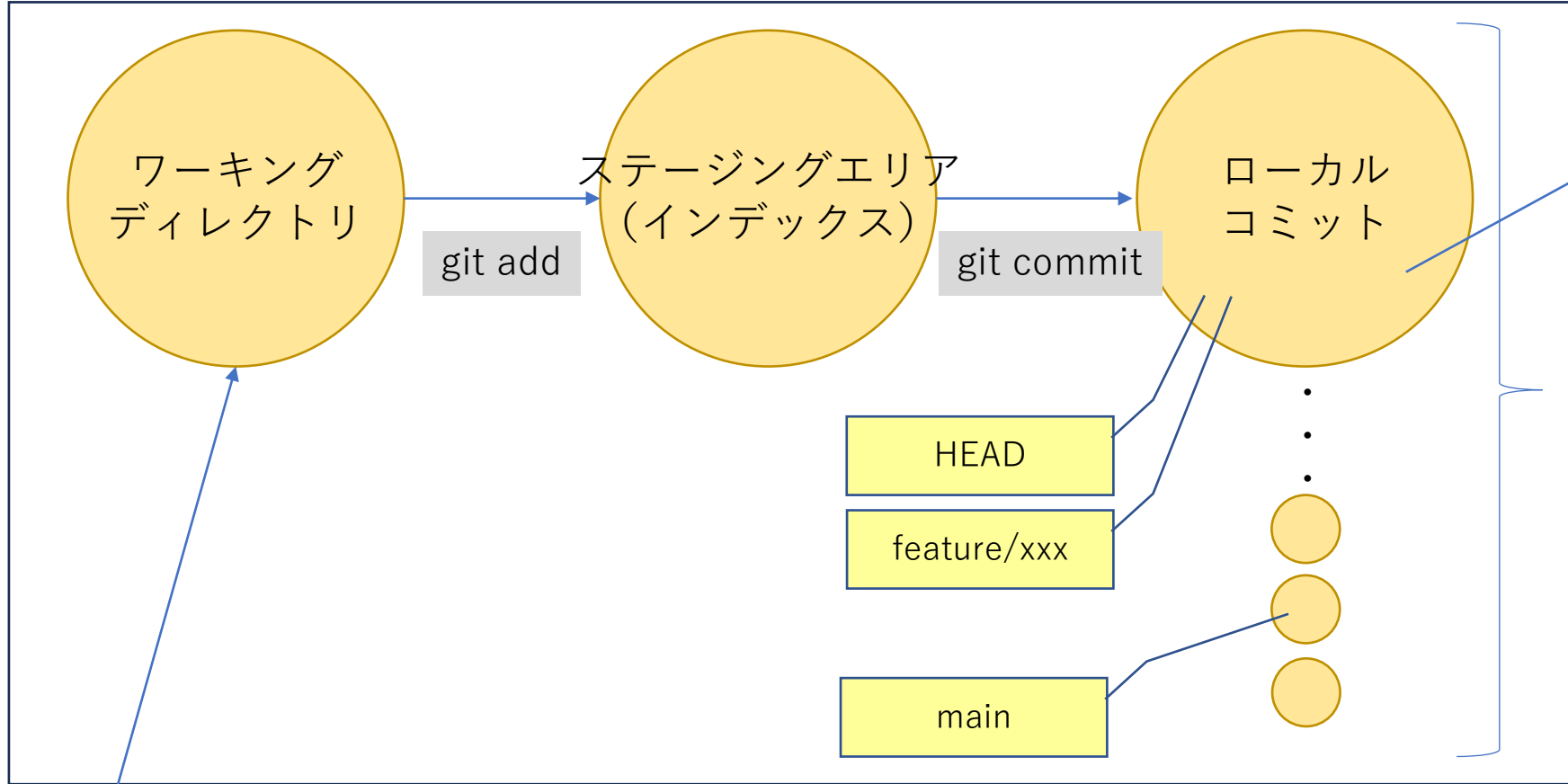
# Gitリポジトリ 概念図

リモートリポジトリ



等

ローカルリポジトリ



git init

git未管理  
ディレクトリ

## 【ターミナルにブランチ名表示, コマンド補間】

- ・ 参考 :

<https://qiita.com/mikan3rd/items/d41a8ca26523f950ea9d>

## 【log一覧 表示項目追加 (git tree エイリアス)】

```
git config --global alias.tree "log --graph --  
pretty=format:'%x09%C(auto) %h %Cgreen %ad %Creset%x09%C(cyan)%an%Creset %x09%C(auto)  
%s %d' --date=format-local:'%Y/%m/%d %H:%M:%S'"
```