Introduction to Development

February 15, 2024

Yu Tokunaga

The primary objective of this document is to facilitate a conceptual comprehension of "What constitutes system development" for individuals lacking IT skills and industry experience. This endeavor is not aimed at presenting efficient methodologies or challenging widely accepted norms; rather, it seeks to methodically consolidate information for those desiring a fundamental grasp of essential points.

Console 3
Coding 17
Algorithm 27
Quality Control 28
Git 30
Web 33
Database 38
Technical Writing 39

Console

コンソール

コンソールとは、OS を搭載したコンピュータに接続されたディスプレイおよびキーボードである。 コンピュータとの対話を可能にするための物理ハードウェア、入出力システムともいえる。

「文字だけの黒い画面」というキーワードで一般に想起される画面は、仮想コンソールである。 これは入出力を仮想的な空間に提供するものであり、この環境は CUI（ Character-based User Interface ）ともいう。 CUI は、我々が日々利用する GUI（ Graphical User Interface ）と対をなす存在であり、文字に基づいたユーザーインターフェースを提供する。

インターフェース

“Interface"という語はハードウェアでもソフトウェアでも登場する。 主に情報産業では、異なる二つのモノ（ 人間や機器 ）を接触させるための境界面として用いられる語である。

ターミナルとシェル

仮想コンソール（CUI）を提供するアプリケーションは一般に*ターミナル*と呼ばれる。Windows におけるコマンドプロンプトや、MacOS における iTerm2 や Warp などがターミナルにあたる。これらは GUI 上で CUI を操作するための窓口になる。

本題となるコンピュータへの命令は*コマンド*という。コマンドを解釈し、実行する仕組みをシェルという。シェルの役割はコンピュータとの対話を実際に担うアプリケーションであり、Bash や zsh、fish など種類が存在する。

シェル

エンジニアでない一般ユーザが OS と対話するためのシェルに、Windows のエクスプローラーなどがある。CUI のシェルと GUI のシェル、操作しやすいユーザ層は当然異なる。

ターミナルとシェル (ii)

要するに、エンジニアがコンピュータへ命令するためには基本的に「ターミナルを起動し（シェルを介して）コマンドを入力」する。また、コマンドを入力する行のことをコマンドラインという。単語が重要なのではなく、この CUI 構成を理解することは今後役に立つ。

コマンド

ここでは、基本的なコマンドを紹介する。細かいオプションまでは解説をしない。なお、ここで使用するコマンドは Windows コマンドプロンプトでは使用できない。

ファイルの一覧化：ls

まずはフォルダの内容を確認したい。そのようなときに打つコマンドである。何回打っても良い．

```
1 ls
```
Command

ディレクトリの移動：cd

今いるフォルダから移動したいときに打つ。

```
1 cd {path} // {移動先のパス}へ移動
2 cd ..     // 1つ上のディレクトリへ移動
```
Command

コマンド (ii)

```
3  cd ~        // ホームへ移動
```

コマンドライン

本書では、コマンドラインへの入力表記において{}を代入記号、//をコメントとする。これらは実際に入力せず、可読性を補助する。

現在地 : pwd

プロンプトに書いてあるディレクトリ名をわざわざ絶対パスで表示する。 あまり使わないが，重大な作業をしているときに使った記憶がある。

```
1  pwd                                                    Command
```

コマンド (iii)

ディレクトリの作成 : mkdir

思ったよりお世話になる。メイクディレクトリと読んでしまうが正解は知らない。知っておいて損はない。

```
1 mkdir {ディレクトリの名前}
```
Command

ファイルの閲覧 : cat

本来は複数のファイルを結合 (concat) して表示するコマンドだが、単一のファイルをのぞき見することで使うことが多い。

```
1 cat {ファイル名}
```
Command

コマンド (iv)

ページャ : less

本来のファイル閲覧用コマンドだが、大体は cat コマンドで足りてしまう。 less を使用した後は、Q キーを押下すると終了する。

```
1 less {ファイル名}
```
Command

容量の確認 : du

「このフォルダは何 MB あるのか」といった疑問を解決する。 ちなみに、-hs オプションを付与すると内訳を非表示にできるため、ファイル数が異常に多いディレクトリで有効である。

```
1 du {フォルダ名}
```
Command

コマンド (v)

検索 : grep

ファイルを検索することは多々ある。 よく使うオプションつきで紹介する。

```
1  grep -rn {path} -e ${text}
```
Command

- r : 再帰的にサブディレクトリを含める
- n : マッチした行番号を表示
- e : 検索する文字列を指定

改名もしくは移動 : mv

便利だが、個人的にリスクがあると思っているコマンドの 1 つである。 2 つめの引数が存在するパスならそこへ 1 つ目の引数に指定されたファイルを移動させる。 存在しないパスなら、 1 つ目の引数に指定されたファイルを 2 つ目の文字列に改名する。

コマンド (vi)

```
1 mv {path} {path} // 改名                    Command
2 mv {path} {rename} // 新しい名前に変更
```

移動先を誤字脱字すると、その文字列が新しい名前となるので注意が必要である。 ただし、-iv オプションを付与すると警告してくれるので必須である。

削除 : rm

mv より危険なコマンドである。 削除をコマンドで実施する場合、取り返しがつかない(=ゴミ箱に移動しない)ため、-iv オプションを付与して警告を有りにする習慣をつけよう。

```
1 rm -iv {対象ファイル}                      Command
```

正規表現

正規表現は、文字列のパターンを表現するための記法である。 これにより、特定の条件を満たす文字列を効果的に検索したり、置換したりすることができる。 基本的な正規表現を以下に示すが，正規表現のあとにコロン:を付与するため注意してほしい。

Character Class：文字

- [abc]: いずれか一つ
  - => a, b, c
- [^abc]: 含まない
  - => d など

Metacharacters：メタ文字

- .: 任意の 1 文字
- \n: 改行コード
- ^: 行の先頭

正規表現 (ii)

- $: 行の末尾
- \d: 半角数字
- \s: 空白
- \S: 空白以外すべて
- \w:半角英数字とアンダースコア
- \l:半角英小文字
- \u:半角英大文字

Quantifiers：量指定

- *: 直前の要素が 0 回以上繰り返し（ ワイルドカードともいう ）
- +: 直前の要素が 1 回以上繰り返し
- ?: 直前の要素が 0 回または 1 回
- {n}: 直前のパターンを n 回繰り返し

Grouping : グルーピング

正規表現 (iii)

- (): 括弧内のパターンを 1 つの要素として扱う

正規表現の例

電話番号は以下の正規表現が適用できる。

```
^(\d{3}-\d{4}-\d{4}|\d{10})$
```

ハイフンを含む形式（例: 123-4567-8901）またはハイフンなしの形式（例: 1234567890）にマッチする。

1. Ctrl+c キーを用いよ．
2. 別のターミナルを起動し，ps，grep で yes コマンドの PID を調べ，kill コマンドを用いよ．
3. top コマンドを用いよ．

Coding

コーディング

プログラムを開発する作業全般をプログラミングといい，特にソースコードを書く工程をコーディングという。 コーディングにおいて、シンタックスとデザインパターンは中核を成す。

シンタックスとは、プログラミング言語の仕様として定められた構文規則を指す。 デザインパターンについては Wiki[1]を引用する。

“ソフトウェア開発におけるデザインパターンまたは設計パターン（英: design pattern）とは、過去のソフトウェア設計者が発見し編み出した設計ノウハウを蓄積し、名前をつけ、再利用しやすいように特定の規約に従ってカタログ化したものである。”

[1]https://w.wiki/rvm

コーディング (ii)

これより、シンタックスに関連する事項を説明する。 通常、伝統的なアプローチでは「ハローワールド関数」から始めることが一般的だが、本書では変数に関する説明から始める。

変数

以下のコードはどのような処理か考えてみよう。

```rust
let x = 4;
```

x は変数、4 は整数リテラル[2]である。 それと同時に、x は 4 を束縛 (bind)する所有者である。 let は変数宣言のキーワードとなっており、以下の構文を想定している。

```rust
let PATTERN = EXPRESSION;
```

整数リテラル 4 単体であっても、右辺は式である。 式が返却する値は 4 であり、左辺のパターン x に当てはめて束縛される。

パターンであるという認識はとても重要で、以下はエラーにならない。

[2] リテラルとはソースコードに生で書かれたデータであり、値そのものである。例えば、4(リテラル)に 5(値)を束縛することはできない。

変数 (ii)

```rust
let (a, b) = (4, 5);
```
Rust

ただし、パターンには 2 つの形式が存在する。上記の_Irrefutable(*論駁不可能*)なパターンと、`let` 式で扱うことができない_*Refutable*(論駁可能)なパターンである。この_Refutability(*論駁可能性*)の概念については、*エラーメッセージで見かけた際に対応できるように、慣れておく必要がある。* Refutable_なパターンはまた別の章で触れる。

変数宣言は、値 4 に別のラベル x を張り替えるような作業ではなく、所有者 x に 4 が束縛されるという認識をもってほしい。

型

Rust 標準のプリミティブ型を表にした。 型の名前に含まれる数字は、ポインタのサイズ(ビット)を指す。

基本データ型	型
符号付き整数	isize, i8, i16, i32, i64
符号無し整数	usize, u8, u16, u32, u64
浮動小数点数	f32, f64
文字(Unicode のスカラー値)	char

特に数値を表す型の許容値は以下となる。

型 (ii)

型	MIN	MAX
i8	−128	127
i16	−32768	32767
i32	-2^{32}	$2^{32} - 1$
i64	-2^{64}	$-2^{64} - 1$
u8	0	255
u16	0	65535
u32	0	2^{32}
u64	0	2^{64}
f32	凡そ-21 億	凡そ+21 億
f64	凡そ-9 京	凡そ+9 京

型 (iii)

isize および usize は実行環境 PC の CPU のビット数によって適切なサイズに変化する。

型推論

初期化の際に評価値の型をチェックするだけでなく、その後にどのような使われ方をしているかを見て推論する。

関数

ハローワールドプログラムを観察する。

```rust
fn main() {
    println!("Hello, world!");
}
```

制御フロー

if

loop

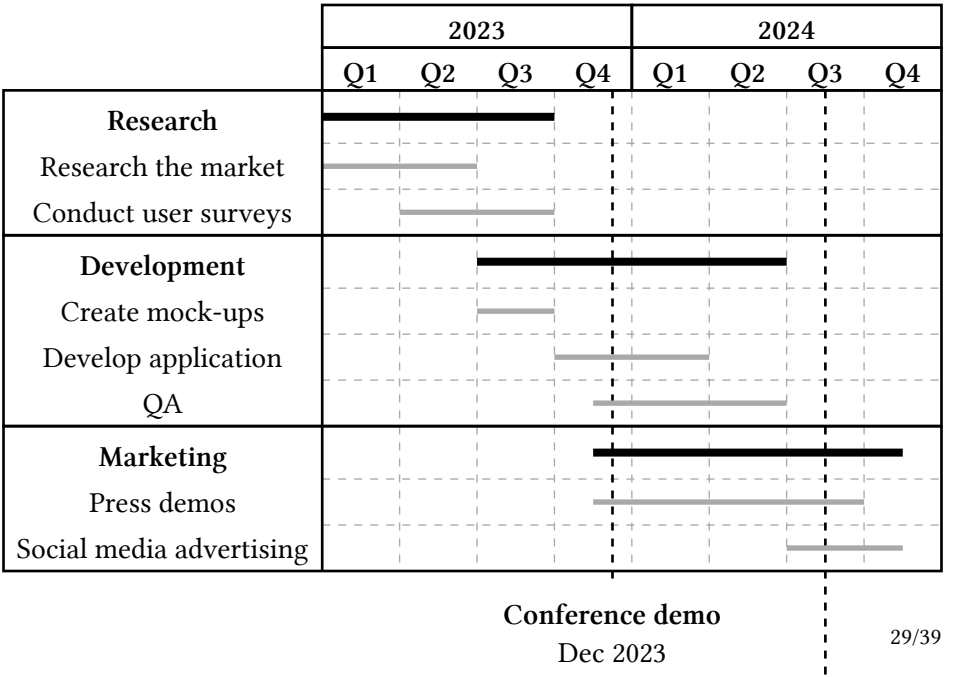
while

for

Algorithm

Quality Control

開発ワークフロー


2023
2024
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Research
Research the market
Conduct user surveys
Development
Create mock-ups
Develop application
QA
Marketing
Press demos
Social media advertising
Conference demo
Dec 2023

Git

分散型バージョン管理システム

Git と GitHub

Git の仕組み

ワーキングディレクトリ

ステージング

リポジトリ

コミット

プッシュ

プル

ブランチ

Web

沿革

本格的にワールドワイドウェブ（ www ）が普及したのは 1995 年頃だった。 当時は静的な HTML（ HyperText Markup Language ）ページが主流だった。 HTML は情報を構造化し、文書の意味や見出し、段落、リストなどを表現していた。 ページのデザインやスタイルは限定的だったが、これが Web の基盤となり、今日の進化したウェブページの基礎となった。

ユーザはインターネットブラウザを介して Web サーバと通信し、HTML を取得している。 さらに、CSS や JavaScript などがページデザインを制御することにより、インタラクティブな体験を得られている。

MPA

ブラウザは Web サーバへ HTTP リクエストを送信する。 Web サーバ内で HTML ページを構築し、JavaScript や CSS とともにに HTTP レスポンスを返却する。 この処理をリクエストのたびに繰り返すため、ページ遷移の度に読み込みが発生する。 この仕組みを Multiple Page

沿革 (ii)

Application(MPA)という。 MPA はページ遷移の速度に問題を抱えていた。

SPA

MPA のデメリットを解消する仕組みが Single Page Application(SPA)である。 MPA は、SPA のように毎回フルの HTML ページを返却するのではなく、差分のみを更新する。 Ajax[3]を使用し、差分情報のリクエストを送信し、サーバサイドは差分箇所を示す Json を返却する。 差分のみの更新は MPA と比べ高速であり、ページ遷移が快適になる。 しかし、初回アクセス時に全ページの描写に必要な JavaScript を取得するため、ローディングコストが発生し、読み込みが重くなる。 代表的なライブラリに `React` がある。

[3]Web ページを表示した状態のまま、新しいページの読込などを伴わずに Web サーバ側と通信し、非同期的に表示内容を変更する技術。

沿革 (iii)

SSG

Static Site Generation(SSG)は、ビルド時に静的な HTML ファイルを生成する。 アプリをビルドするたびに、全てのページが作成される。 ユーザーが Web サイトにアクセスするとこれらのファイルをロードされ、サーバーは余計な作業をする必要がなくなる。 ホームページやブログなどの更新頻度が少ない静的コンテンツに適切なアプローチである。 代表的なフレームワークに `Next.js` がある。

SSR

SSR(Server Side Rendering)とは、初期表示速度の問題を解決するために、サーバーサイドでページの内容をレンダリングするアプローチである。 ユーザーは最初のページロードで完全なページコンテンツを受け取る。 一度ページを読み込むと、その後のデータの取得や更新は最小限のリソースで行えるため、ネットワークトラフィックやサーバーロードを軽減できる。 SSR ページはリクエストごとにサーバー上で生成される

沿革 (iv)

ため、常に最新のデータを表示する。 通販サイトや SNS プラットフォームのような、パーソナライズされた (動的) コンテンツを持つアプリケーションに適切である。 代表的なフレームワークに `Remix` がある。

Database

Technical Writing