

情報実験・第3回 (2024/05/10)

---

# 最低限Unix(Linux) II

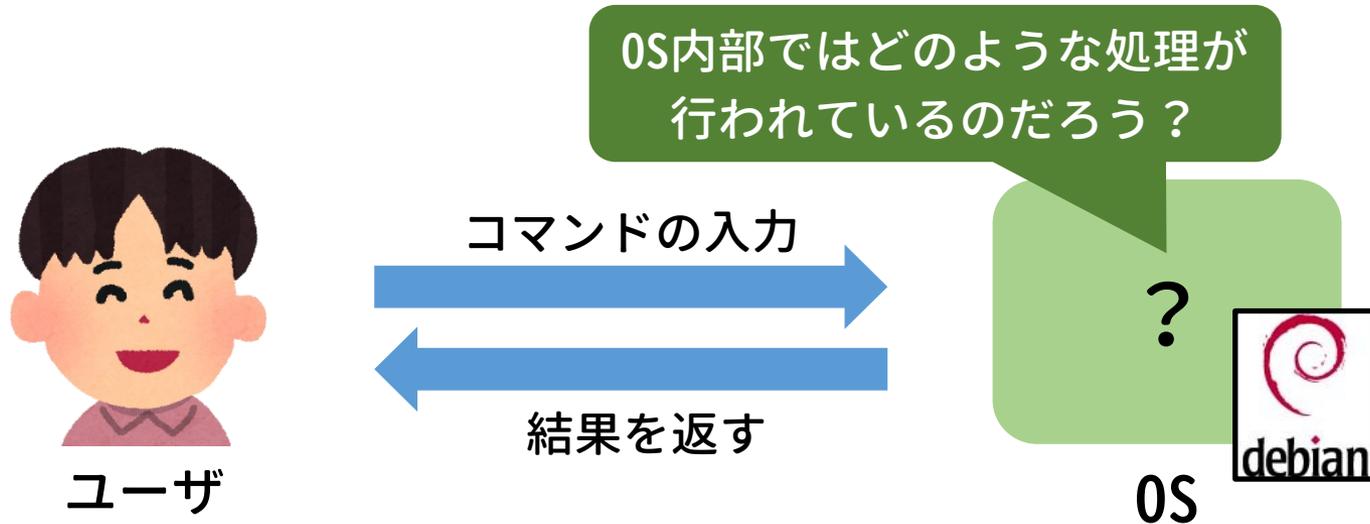
## ～シェル・テキストエディタ～

北海道大学大学院 理学院 宇宙理学専攻  
修士課程 2年  
高橋 聖輝 / Takahashi Masaki



# 本日の内容

## ■ コマンドが実行される仕組みについて



## ■ テキストエディタについて

- テキストエディタとは？
- viの使い方

# 目次

---

1. カーネルとシェル
2. シェルスクリプト
3. テキストエディタ

# 1. カーネルとシェル

---

# 復習：OS(Operating System)

## ■OSとは？

- 計算機を管理，操作するための基本ソフトウェア
- アプリケーションソフトウェアとハードウェアの仲介役



# OSの大まかな構造

---

カーネル(Kernel:核)とシェル(Shell:殻)の  
二段階構造



# カーネル

## カーネル (Kernel, 核)

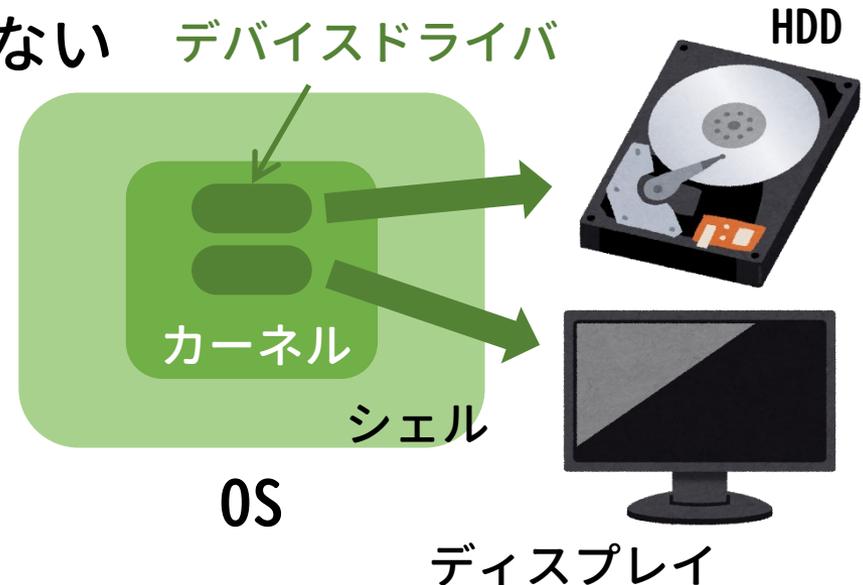
- OSの中核をなすソフトウェア
- ソフトウェアの要求に対して、デバイスドライバを介して  
(第五回参照) 必要なハードウェアを制御する  
e.g. CPU に計算を実行させる, HDD にデータを保存する

ユーザはカーネルを直接操作できない デバイスドライバ

→ ユーザとカーネルを仲介する  
ソフトウェアが必要



シェル

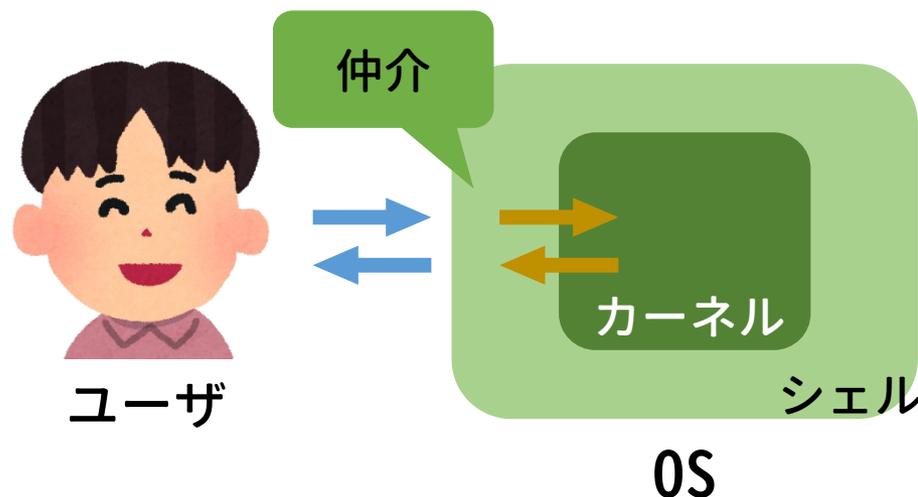


# シェル

## シェル (Shell, 殻)

- ユーザとカーネルを仲介するソフトウェア
- ユーザはシェルを通して計算機に作業を要求する
- ユーザインタフェース (UI, ユーザとの境界部分) を実装

GUI: Graphical User Interface  
CUI: Character User Interface



# ユーザインタフェースの例：GUI

The image shows a composite screenshot of a presentation software interface. The main window displays a slide titled "カーネル" (Kernel) with the following text:

カーネル (Kernel, 核)

- OSの中核をなすソフトウェア.
- ソフトウェアの要求に対して、デバイスドライバ (第五回参照) e.g. CPUに計算

ユーザはカーネルできない。  
→ ユーザとカーネルのソフトウェア

ノートを入力

3.54564, 0.834551

The Gnuplot window shows a graph of  $\sin(x)/x$  with a peak at  $x=0$  and a value of approximately 1.0. The x-axis ranges from -15 to 15, and the y-axis ranges from -0.4 to 1.0.

The web browser window displays a scientific article from the Journal of Geology of Japan, Vol. 123, No. 9, p. 733-745, September 2017. The article title is "変成岩組織と鉱物組成累帯構造からの情報抽出：フォーワードモデルと逆解析" (Information extraction from metamorphic rock textures and compositional zoning of minerals: Forward models and inversion analyses). The authors are Sushi Okamoto\*1 and Tetsu Kuwatani\*2, \*3. The abstract discusses the challenges of understanding metamorphic processes and the use of forward modeling and inversion analysis.

# ユーザインタフェースの例：CUI

```
Debian GNU/Linux 12 tty1
```

```
joho01 login: tmasa
```

```
Password:
```

```
tmasa@joho01: ~$ ls -F
```

```
240322_Arc/ 240422_Arc/ IMG_1377.JPG Bthesis.bak/
```

```
tmasa@joho01: ~$ |
```

# GUI と CUI の特徴

---

## GUI の特徴

- マウス・タッチパネル等を使って直感的に作業できる
- 計算機への負荷が大きい（CUI よりも計算機の動作が複雑）

## CUI の特徴

- コマンドを覚えればキーボードだけで何でもできる
- 計算機への負荷が小さい
  - サーバ業務，トラブル対処に強い
- 単純な繰り返し作業に向く（本日の課題にも関連）

# シェルの基本的機能

---

- コマンドインタプリタ  
(Command Interpreter)
- 環境設定

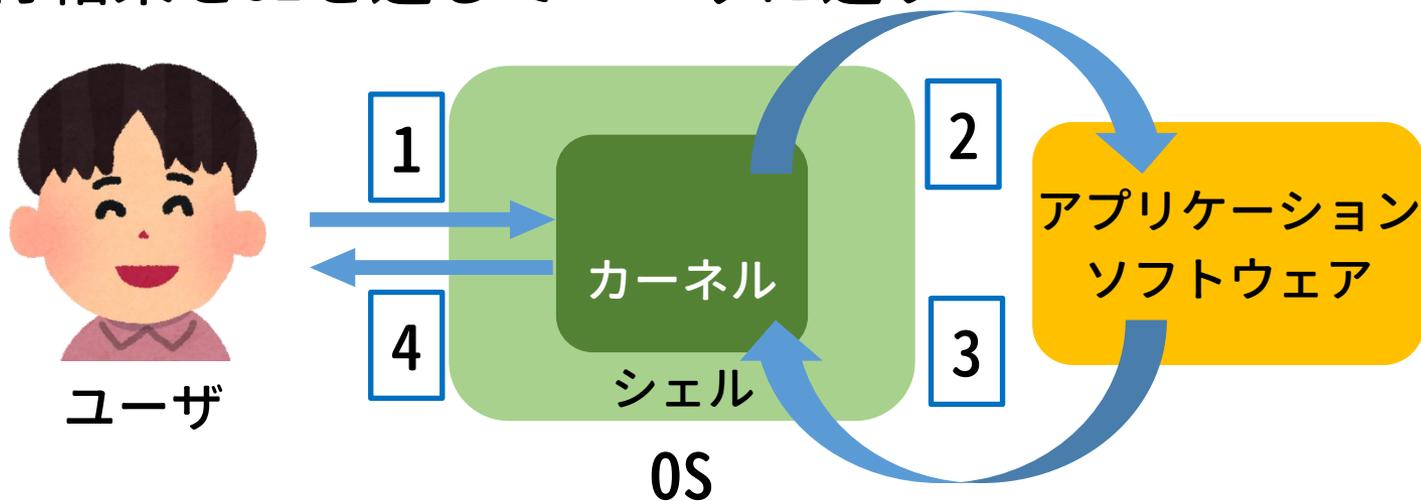
# シェルの基本的機能

---

- コマンドインタプリタ  
(Command Interpreter)
- 環境設定

# コマンドインタプリタ

1. UIを通してユーザのコマンドを受け取り、カーネルに実行を要求する
2. 適切なアプリケーションソフトウェアに引き渡す
3. アプリケーションソフトウェアから実行結果を受け取る
4. 実行結果をUIを通してユーザに返す



# コマンドインタプリタの仕事



ユーザ



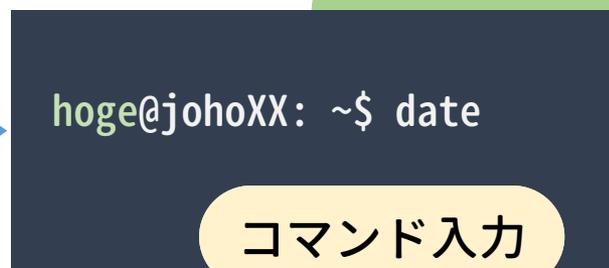
コマンドを待ち受ける (これも仕事の一つ)

# コマンドインタプリタの仕事

今何時ですか？



ユーザ



シェル

カーネル

アプリケーション  
ソフトウェア

ここでのシェルの機能

- 文字列 `date` を受け取る
- `date` というコマンドを探し出し，カーネルに実行を依頼

# コマンドインタプリタの仕事

13時15分かあ  
3限寝坊したわw



ユーザ

```
hoge@johoXX: ~$ date  
Fri May 10 13:15:24 JST 2022
```

日時の表示

シェル

カーネル

アプリケーション  
ソフトウェア

ここでのシェルの機能

- カーネルから date コマンドの結果を受け取る
- 結果を UI に表示する

# シェルの基本的機能

---

- コマンドインタプリタ  
(Command Interpreter)
- **環境設定**

# 環境設定

---

## 環境

- アプリケーションソフトウェア間で共用される設定情報  
(e.g. 言語)
- 各アプリケーションソフトウェアはシェルから与えられた環境下で動く

## 環境変数

- 設定内容を格納する変数  
変数 `LC_ALL` に代入されている値の例：`ja_JP.UTF-8`
- 起動時に自動設定されるが、手動で書き換えることもできる  
例：`export LC_ALL=ja_JP.UTF-8`  
例：`export LC_ALL=en_US.UTF-8`

# 環境設定

---

```
tmasa@joho01:~$ echo $LC_ALL 変数 LC_ALL に代入されている値の確認  
en_US.UTF-8  
tmasa@joho01:~$
```

# 環境設定

---

```
tmasa@joho01:~$ echo $LC_ALL
```

```
en_US.UTF-8
```

```
tmasa@joho01:~$ date
```

```
Fri May 10 12:44:46 JST 2024
```

現在の日時が英語で表示される

```
tmasa@joho01:~$
```

# 環境設定

---

```
tmasa@joho01:~$ echo $LC_ALL
```

```
en_US.UTF-8
```

```
tmasa@joho01:~$ date
```

```
Fri May 10 12:44:46 JST 2024
```

```
tmasa@joho01:~$ export LC_ALL=ja_JP.UTF-8
```

変数の値を変更

```
tmasa@joho01:~$
```

# 環境設定

---

```
tmasa@joho01:~$ echo $LC_ALL
```

```
en_US.UTF-8
```

```
tmasa@joho01:~$ date
```

```
Fri May 10 12:44:46 JST 2024
```

```
tmasa@joho01:~$ export LC_ALL=ja_JP.UTF-8
```

```
tmasa@joho01:~$ echo $LC_ALL 変数 LC_ALL に代入されている値の確認
```

```
ja_JP.UTF-8
```

```
tmasa@joho01:~$
```

# 環境設定

---

```
tmasa@joho01:~$ echo $LC_ALL
```

```
en_US.UTF-8
```

```
tmasa@joho01:~$ date
```

```
Fri May 10 12:44:46 JST 2024
```

```
tmasa@joho01:~$ export LC_ALL=ja_JP.UTF-8
```

```
tmasa@joho01:~$ echo $LC_ALL
```

```
ja_JP.UTF-8
```

```
tmasa@joho01:~$ date
```

```
2024年 5月 10日 金曜日 12:45:22 JST 現在の日時が日本語で表示される
```

```
tmasa@joho01:~$
```

# 環境設定

```
tmasa@joho01:~$ echo $LC_ALL
```

```
en_US.UTF-8
```

```
tmasa@joho01:~$ date
```

```
Fri May 10 12:44:46 JST 2024
```

```
tmasa@joho01:~$ export LC_ALL=ja_JP.UTF-8
```

```
tmasa@joho01:~$ echo $LC_ALL
```

```
ja_JP.UTF-8
```

```
tmasa@joho01:~$ date
```

```
2024年 5月 10日 金曜日 12:45:22 JST
```

```
tmasa@joho01:~$
```

半角スペースを入れない！

# シェルの種類

---

多様なシェルが存在

sh, bash, csh, dash, tcsh, zsh ...

INEX では **bash** を使う

色々便利な機能を持った標準的なシェル

ヒストリ機能, 補完機能,  
リダイレクト...

# シェルの種類

---

多様なシェルが存在

sh, bash, csh, dash, tcsh, zsh ...

INEX では **bash** を使う

色々便利な機能を持った標準的なシェル

ヒストリ機能, 補完機能,  
リダイレクト

詳しくは実技編で！！

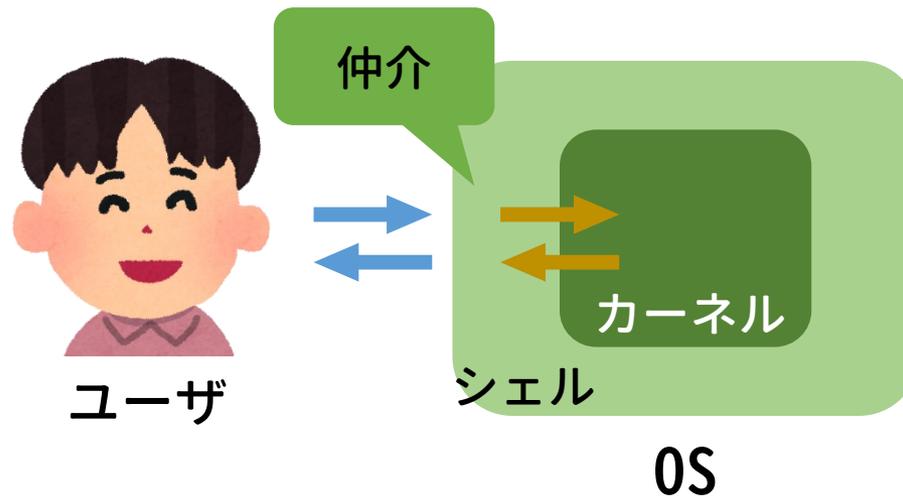
# 前半のまとめ

## カーネル

- OSの中核のソフトウェア
- ソフトウェアの要求に対して必要なハードウェアを制御

## シェル

- ユーザとカーネルを仲介するソフトウェア
- ユーザインタフェースの提供（ユーザと接する境界）
- ユーザはシェルを通して計算機に作業を要求



# 前半のまとめ

---

## シェルの基本的機能

- コマンドインタプリタの提供
- 環境の設定
  - アプリケーションソフトウェア間で共用される情報を設定できる

# 前半の実習では…

---

## シェルに慣れる

- シェル（特にbash）の各種機能を試してみよう！

## 2. シェルスクリプト

---

# シェルスクリプト

---

## シェルスクリプト

- … コマンドを実行順に並べて記述したファイル  
(script = 台本)

台本を読むように  
コマンドを連続して実行できる

### 利用するメリット

- 繰り返し作業の手間を省ける
  - 制御構造を利用したプログラミングが可能
- 人為的ミスを防げる
- 似たような作業をするときに再利用ができる
  - スクリプトファイルの資源化が可能

# 制御構造 ～アルゴリズムの基本～

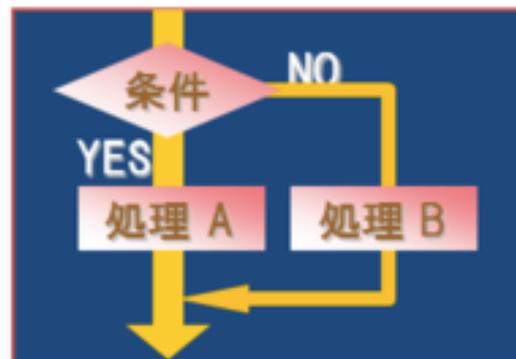
## 順次構造

一方向に順序立てて処理を行う構造



## 選択構造

条件に応じて処理を分岐する構造



## 反復構造

同じ処理を反復する構造



これらの組み合わせで  
色々な作業が可能となる！

# シェルスクリプトの具体例 ①

## 要求

- source.txt のバックアップを取る
- 大切なファイルを別名でも保存する
- 念のため元のファイルは残す

## 手法

- 日付を変数に格納
- ファイル名を backup\_日付.txt としてコピー

```
1 #!/bin/bash↵
2 ↵
3 NICHU=$(date _ '+%Y-%m-%d') ↵
4 ↵
5 cp _source.txt _backup_${NICHU}.txt↵
```

# シェルスクリプトの具体例 ②

## 要求

- 通し番号が付くファイルの作成
- 1-50, 51-100 で名前の付け方を変える

## 手法

- 通し番号の変数を利用する
- 選択構造 (if) と反復構造 (while) を組み合わせる

```
1 #!/bin/bash↵
2 ↵
3 num=1↵
4 ↵
5 while [_$num_ -le 100 ]↵
6 do↵
7   if [_$num_ -le 50 ]; then↵
8     echo "${num}!!!" > small_${num}.txt↵
9   elif [_$num_ -ge 51 ]; then↵
10    echo "${num}!!!" > large_${num}.txt↵
11   fi↵
12   num=$(( $num + 1 ))↵
13 done↵
```

### 3. テキストエディタ

---

# テキストエディタとは？

---

- テキストファイル（テキストデータのみからなるファイル）の編集を目的とするアプリケーションソフトウェア
  - プログラム編集用のソフトウェアが起源
- 通常の記事からプログラム、各種設定ファイルの作成、編集まで幅広く使える
- 種類が豊富にある  
(例：vi, vim, emacs, nano, メモ帳, Terapad, 秀丸エディタ, VS Code など…)

本日の実習では **vi** を用いる

# vi ～困ったときに頼れるアイツ～

---

テキストエディタの一つ

Unix黎明期から使われている由緒正しいエディタ

## 特徴

- 動作が軽快
- どのLinuxでもほぼ確実にインストールされている
  - トラブル時に役立つ → 管理者にとって必修のエディタ
- 操作方法が**かなり独特**で、慣れが必要

# viの操作概略

シェル

\$ vi [filename]

:wq, :q!

Vi(コマンドモード)

a, A, i, I, o, O キー

ESC キー

Vi(挿入モード)

# viの操作概略

シェル

\$ vi [filename]

:wq, :q!

困ったらEscキーで  
コマンドモードへ!!!

Vi(挿入モード)

# 後半のまとめ

---

## シェルスクリプト

- コマンドを実行順に並べて記述したファイル
- 制御構造を利用したプログラミングが可能

## テキストエディタ

- テキストファイルを編集するためのソフトウェア
- 通常の文書，プログラム，設定ファイルの作成/編集が可能

## vi

- Linux に標準的にインストールされているテキストエディタ
- 動作が軽快で，いざという時に必須となるツール
- コマンドモードと挿入モードがある

# 後半の実習では…

---

## viを使えるようにする

- 最低限のテキスト編集技術を身につけよう！

## シェルスクリプトを書いてみる

- 煩雑な作業をスクリプトを書くことで効率化しよう！