

# 情報リテラシ 第二

2023年度2Q 5c/6c(IL2) 木曜日

担当：地引

TA：増井

# 本日の講義内容

---

- コンピュータの仕組み
  - アーキテクチャ, 基本ソフトウェア (OS)

- ファイル システム
- コマンド ラインの利用

剥き出しのコンピュータを  
直接触るイメージです。

# コンピュータの構造

# 最初の計算機

- 階差機関 (Difference Engine) : 19 世紀前半
  - 多項式関数では、有限次の階差が一定値になることを利用
    - 2 次関数: 1, 4, 9, 16, 25, 36, 49 → 3, 5, 7, 9, 11, 13 → 2, 2, 2, 2, 2
  - 加算のみで計算が可能 (階差を逆算する) → 歯車の利用
    - チャールズ・バベッジ: 5 桁の数値を 2 次階差まで演算できる装置を試作
    - 歯車の摩擦が大き過ぎて精度を上げられなかった (蒸気機関を適用?)。
- 解析機関 (Analytical Engine) : コンピュータの先祖 / 構想のみ
  - 階差機関は多項式のみで三角関数や円周率を計算できない。
  - **計算対象の数値**と**計算処理の手順**とを分け、**状況に応じて外部から指示**する必要性に思い至る。
    - 割り算では被除数から除数を何回引けるかを数えるが、引いた値が+の間は引き算を繰り返し、-になったら次の手順に進む (計算を終了する)。
    - パンチカードを用いて**途中結果の格納 / 次の手順の指示**を行なう。

# 階差機関の原理

階差機関による  $f(x) = 2x^2 + x - 1$  の計算例 ( $x = 5$  の場合を計算)

$x$	$f(x)$	$d1(x) = f(x+1) - f(x)$	$d2(x) = d1(x+1) - d1(x)$
0	-1	3	4
1	2	7	4
2	9	11	4
3	20	15	4
4	35	19	
5	54		

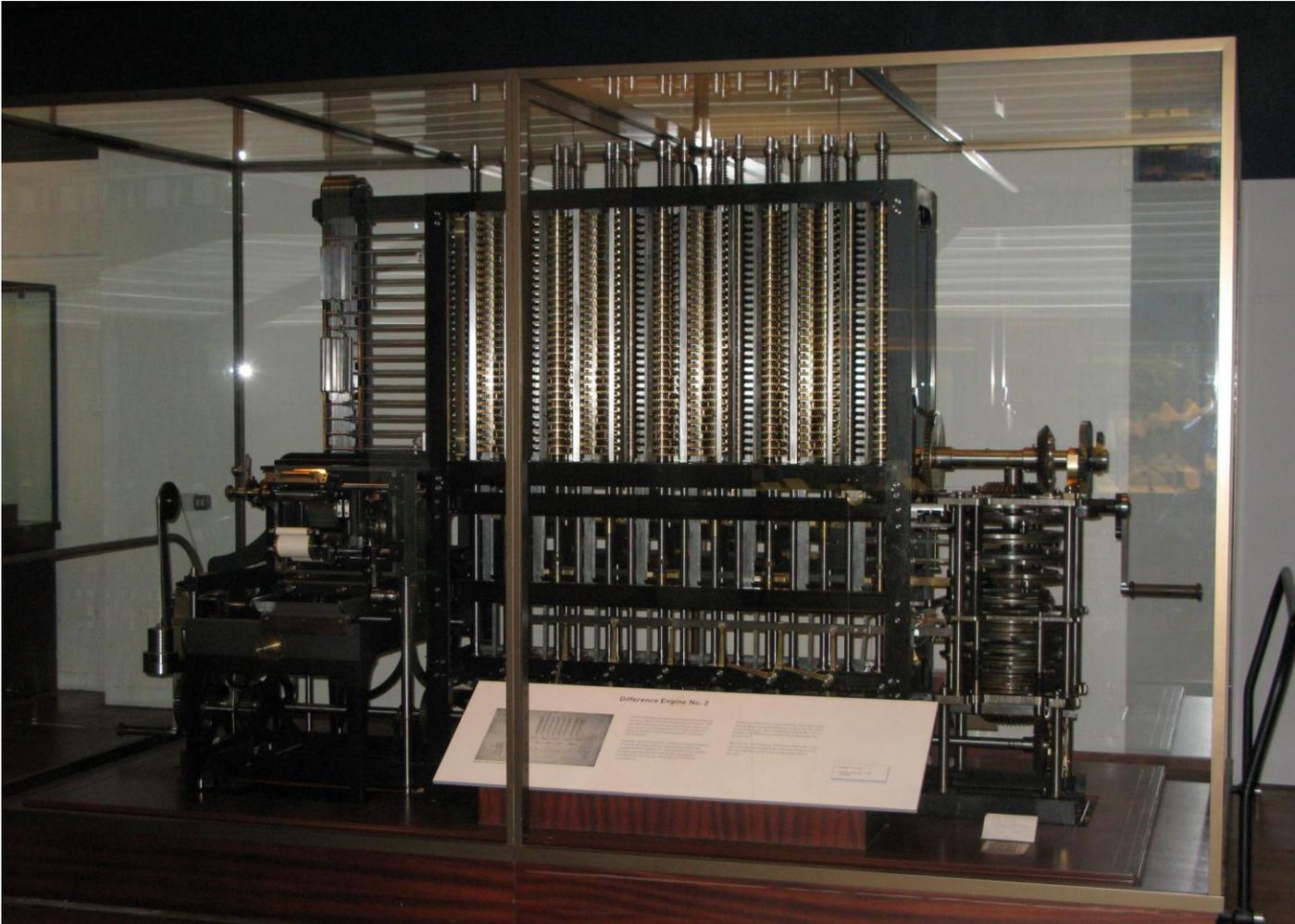
① まずは、各階差が等しくなるまで  $x$  を増やしながら分解

②  $N+1$  次階差を用いて  $N$  次階差を計算

③ 以後、1 次階差を基準にして、順次、関数値の計算を繰り返す

利用するのは足し算のみ  
⇒ 全て歯車で実装できる

# 階差機関の実装例



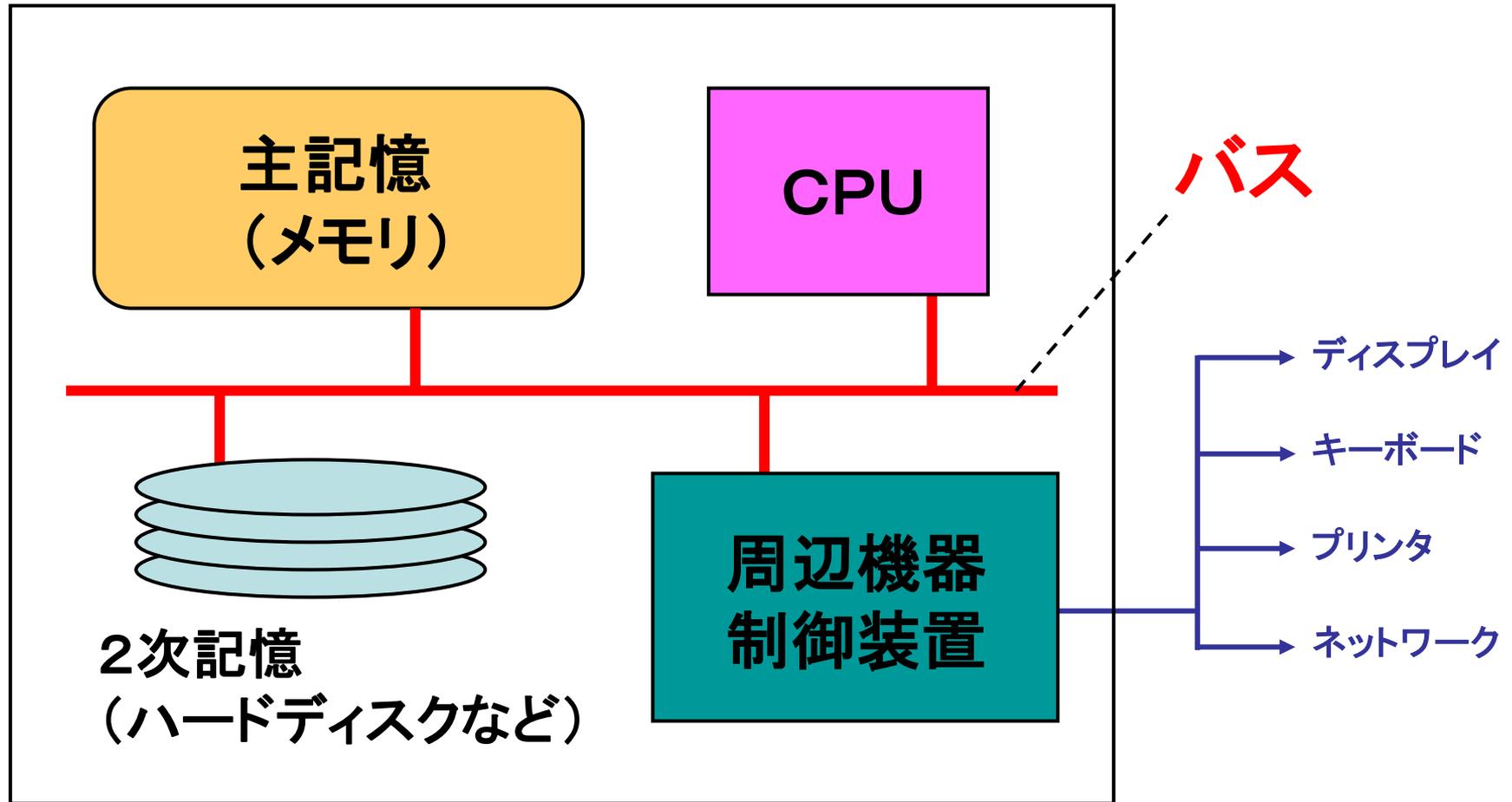
ロンドン科学博物館にある階差機関の実装例 ([Photo by Geni, CC BY-SA 4.0](#))

# 計算機の本質的な特徴

---

- 電卓と表計算を比べてみよう。
  - どちらも、数値計算をするものであるが・・・
- 電卓
  - 計算する毎に数値を入力する必要あり。
  - 1万店舗の売り上げを計算後、誤りが発覚した場合は計算(入力)し直す必要あり？(ぞっとするよね)
- 表計算
  - **計算方法と計算対象(数値)を分ける**ことができる。
    - 数値の入力(セルへの記入)と数値の(セルの)計算方法は別に設定する。
  - **計算対象を保存**しておくことができる。
    - 計算のやり直しや計算結果の流用が可能

# コンピュータの構造



コンピュータを高速化するために重要な部分は？  
⇒ 構造の特徴を見抜く力を身に付けよう

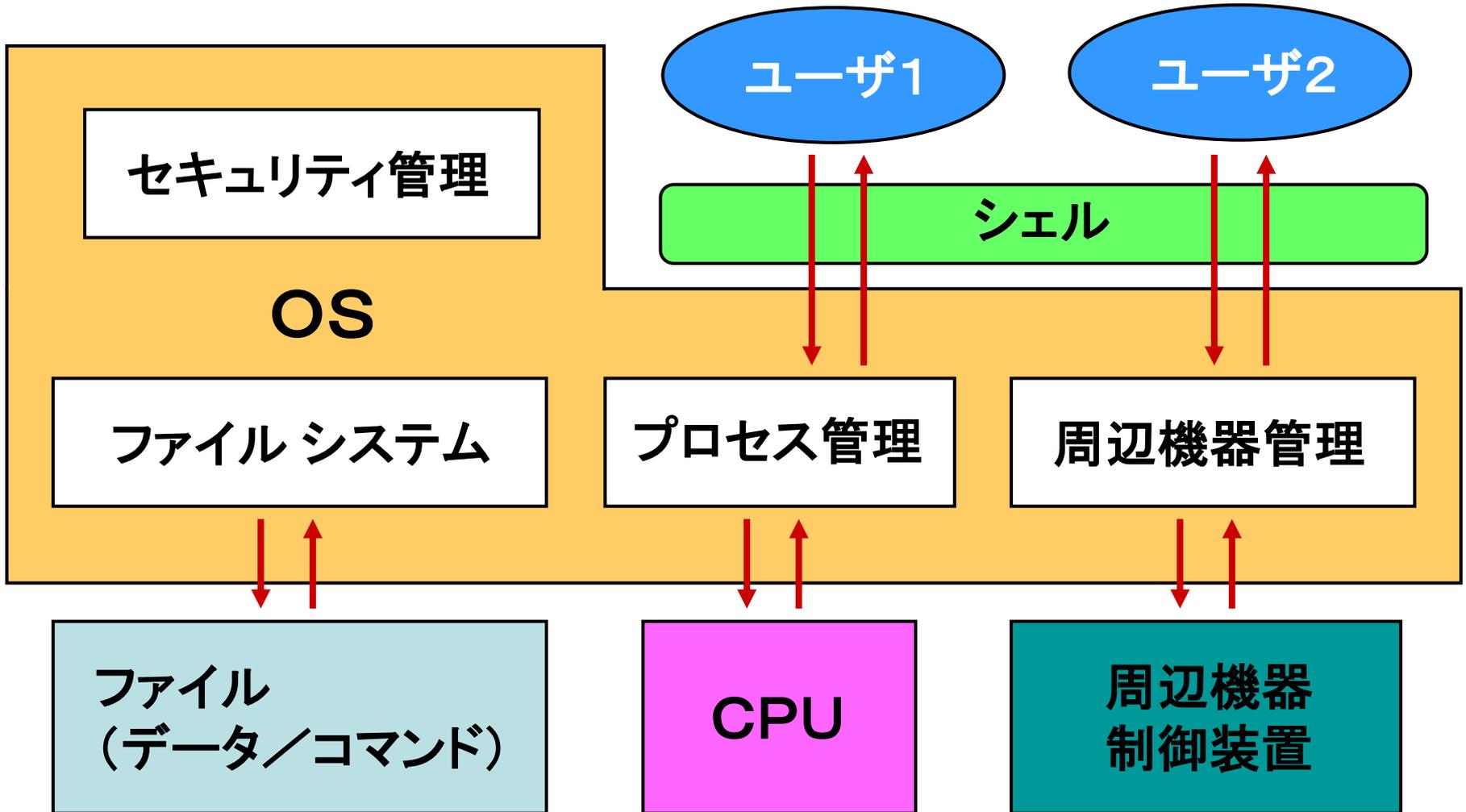
# コンピュータにおける処理

---

- 各種ハードウェアの制御
- 各種データの管理
- 複数の利用者(ユーザ)間での排他制御
  - ユーザAの利用中はユーザBが利用できないという事象をできるだけ減らす。
  - ユーザAの処理がユーザBに予期せぬ副作用(≒妨害?)を及ぼすことをなくす。
- これらの処理を、できるだけ目に見えない形で利用者に負担をかけずにやりたい。

# OS の導入

OS: Operating System (基本ソフトウェア)



# OS の種類

---

- 大型コンピュータ（スパコン／メインフレーム）
  - 専用高速 OS, 一部 UNIX
- サーバ／個人用（ワークステーション／パソコン）
  - UNIX (Solaris, IRIX, FreeBSD, NetBSD, Linux)
  - Windows (Microsoft 社製, パソコン OS シェアNo.1)
  - Mac OS (Apple 社製, パソコン OS の原点)
- 携帯機器／組込み機器（携帯電話／家電）
  - 専用軽量 OS (Android, iOS, Symbian)

# ファイルの管理 (ファイル システム)

# ファイルとフォルダ（ディレクトリ）

---

- ファイル（≡ 書類）
  - データを管理する基本単位 ⇒ サイズは自由
  - 所有者／所有グループ／許可情報を持つ
  - 許可情報：所有者, グループ, 他人毎に  
読み, 書き, 実行の許可を表わす
- フォルダ or ディレクトリ（≡ 書類入れ）
  - 関連するファイルを集めた仮想的な入れ物
  - 別のフォルダを入れ子状に格納できる
  - フォルダの階層を辿ることでファイルを一意に指定できる ⇒ パス名

# ファイルの許可情報

- ファイルの属性 (ls -l で確認)
  - 所有者 / 所有グループ / 許可情報を持つ
  - 許可情報: **所有者**, **グループ**, **他人** 毎に  
読み, 書き, 実行の許可を表わす
- 属性の変更
  - chmod **□□□** ファイル名
  - □ → 読み = 4, 書き = 2, 実行 = 1 (足し算)
    - $a_0 \sim a_{n-1}$  の項目を  $2^0 \sim 2^{n-1}$  に対応させる。
    - $0 \sim 2^n - 1$  の数字で、 $a_0 \sim a_{n-1}$  のあらゆる組合せを表現できる。
  - chmod 755 abc
    - ファイル abc の所有者は、読み / 書き / 実行ができる
    - 所有者と同じグループのメンバは、読み / 実行ができる
    - 全くの他人は、読み / 実行ができる

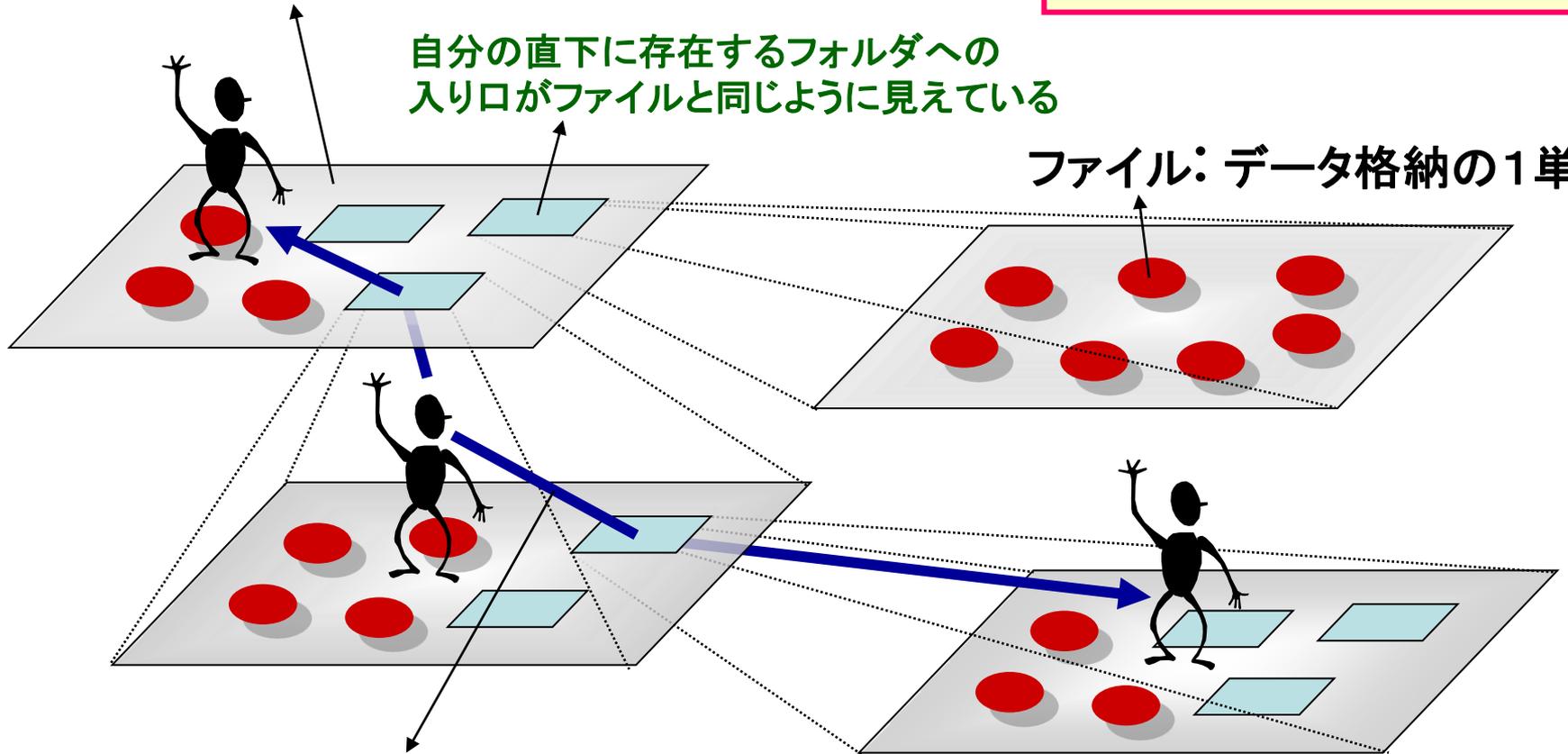
# ファイルとフォルダのイメージ

フォルダ：複数のファイルを格納する入れ物

ファイルを管理する仕組みを  
**ファイルシステム**と呼びます。

自分の直下に存在するフォルダへの  
入り口がファイルと同じように見えている

ファイル：データ格納の1単位

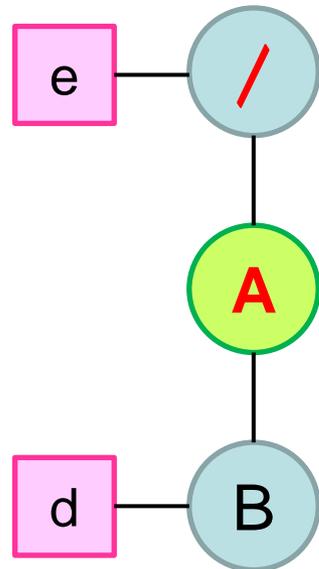


フォルダは入れ子構造になっており、  
上/下のフォルダ間を行き来できる。



# フォルダの階層とその表現 (2)

- **頂上フォルダを基点**にする場合



頂上のフォルダ(「**ルートフォルダ**」と呼ぶ)の名前は“/”と決められている。

カレントフォルダ

**ルートフォルダの名前とフォルダの区切り記号が同じ**なので、ややこしい…  
⇒ 変則性については次のスライド

カレントフォルダが A だとしても、  
ルートフォルダを基点にファイル d を指定する場合: **/A/B/d**  
同じく、ファイル e を指定する場合: **/e**

**ルートフォルダを基点に指定する場合は、カレントフォルダに関係なく、ルートフォルダからフォルダ名を“/”で連結**

# フォルダの階層とその表現 (3)

- 基点となるフォルダにより、2通りの表現がある。
    - カレントフォルダが基点 → “**相対パス**” と呼ぶ。
    - ルートフォルダが基点 → “**絶対パス**” と呼ぶ。
    - 先頭に “/” があるかどうかで両者を判別できる (但し、下記に注意)。
  - 例外!! {
    - ルートフォルダの名前は必ず “/” ⇒ 区切り記号と同じ…
    - 絶対パスの場合、本来ならば “//A/B/d” と表現したいが、“/” は二つ並べない (一つ省略) …… この例外が少々変則的
- 余談: コンピュータ黎明期、//A/… の A はフォルダ名でなくサーバ名を表わすとしたシステムもありました。
- 相対パスと絶対パスの比較 (理解を深めよう)
    - ルートフォルダ内にファイル e が存在する場合の表現 2通り:
      - カレントフォルダをルートフォルダとした場合の相対パス = “**e**”
      - 絶対パス = “**/e**”

“e” と “/e” の違いを理解できれば、一人前

# ファイルの種類

---

- 通常ファイル
  - データ／文書など
- シンボリック ファイル
  - 別の場所にあるファイルをその場所にあるファイルのように見せる
- 特殊ファイル
  - 周辺機器をファイルのように扱える
- ファイル拡張子で中身を表わす
  - あらゆるデータが規定されているわけではない

# ファイルの拡張子

---

## ファイル拡張子の例

- html, htm
  - ホームページの内容が記述されたファイル
- tex
  - Tex, LaTeX (文書処理ツール)で処理するための文書ファイル(≒ 原文)
- c
  - c 言語プログラム ファイル
- pdf
  - 普遍性のある電子文書フォーマット

以下は、参考です。

最終的に、皆さんが習得すべきスキルは、後ほど説明する  
コマンドラインからパス名でファイルを指定できるスキルです。

# Windows でファイル システムを触る

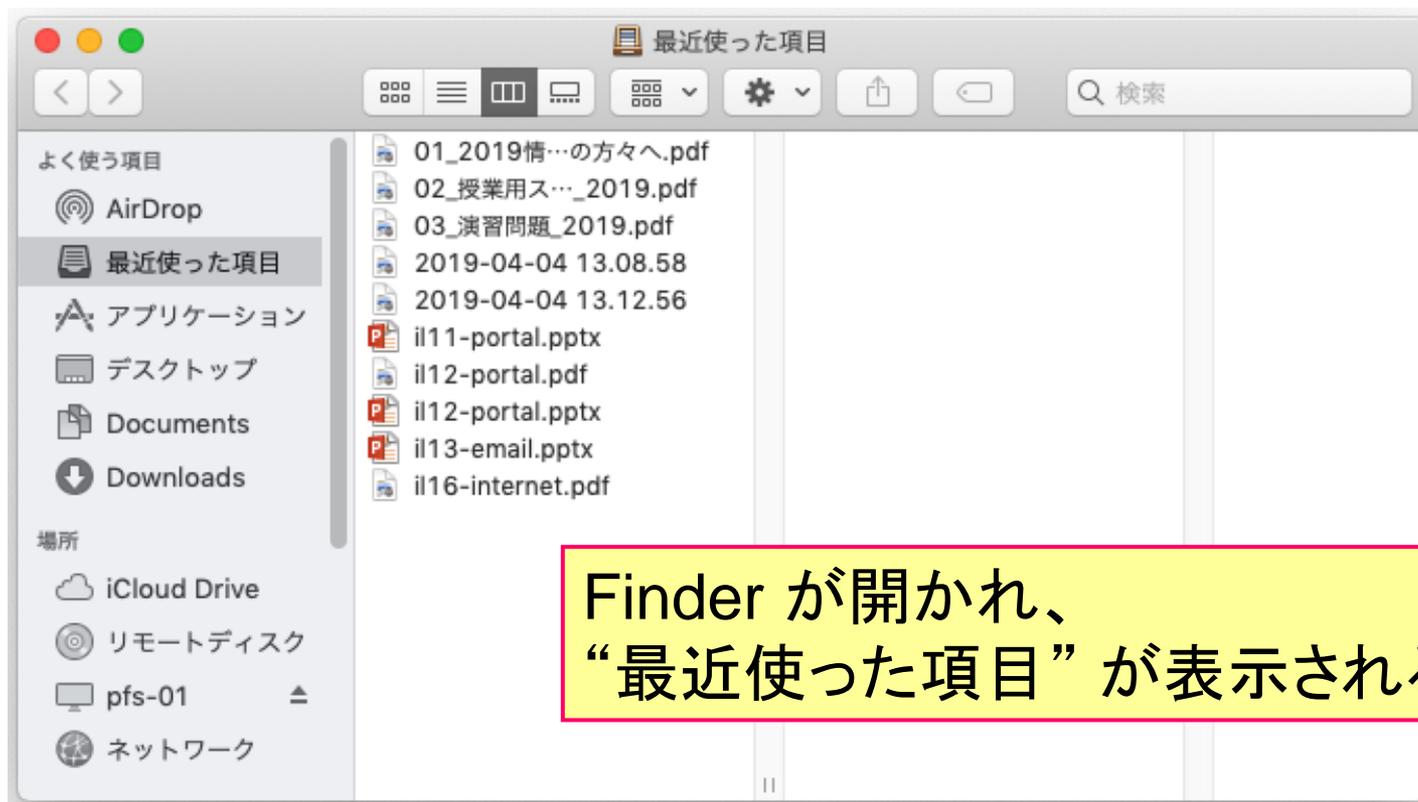
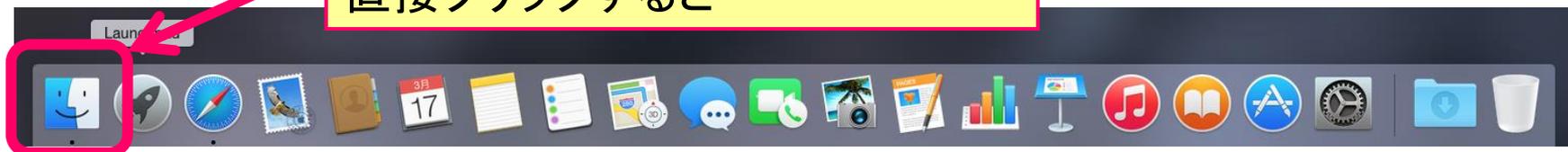
Windows では、左下の“スタート”以下より、“Windows システム ツール”内にある“エクスプローラ”を用いてファイルの操作をします。

名前	更新日時	種類	サイズ
acadres	2017/12/26 15:14	ファイル フォルダ	
AcroApp	2017/12/26 15:14	ファイル フォルダ	
AcroCEF	2017/12/26 15:23	ファイル フォルダ	
ActiveX	2017/12/26 15:23	ファイル フォルダ	
Air	2017/12/26 15:23	ファイル フォルダ	
AMT	2017/12/26 15:14	ファイル フォルダ	
ant_assets	2017/12/26 15:14	ファイル フォルダ	
Browser	2017/12/26 15:23	ファイル フォルダ	
Color Swatches	2017/12/26 15:14	ファイル フォルダ	
Data	2017/12/26 15:23	ファイル フォルダ	
Doc Settings	2017/12/26 15:14	ファイル フォルダ	
DocTemplates	2017/12/26 15:14	ファイル フォルダ	
FileInfo	2017/12/26 15:14	ファイル フォルダ	

エクスプローラでは、左側にファイル システムの階層構造が、右側に左側で指定したフォルダの中身が表示されます。  
[表示] → [ナビゲーション ウィンドウ]にある、“開いているフォルダーまで展開”にチェックを入れると、左側での移動と右側での移動が同期されるので、便利です。

# Finder からファイル システムを触る (1)

Dock 上にある Finder アイコンを  
直接クリックすると…



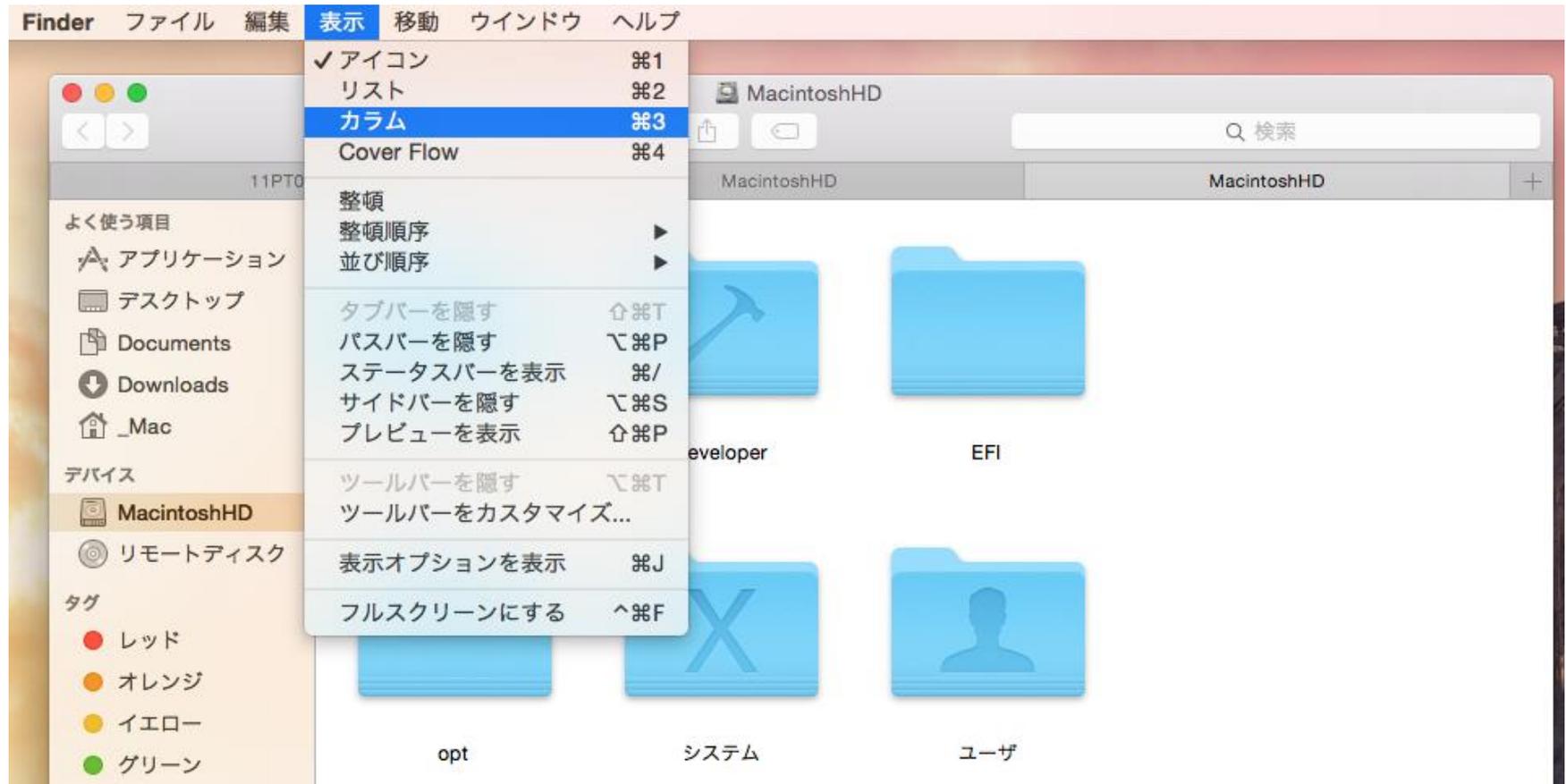
# Dock から Finder を起動できない場合



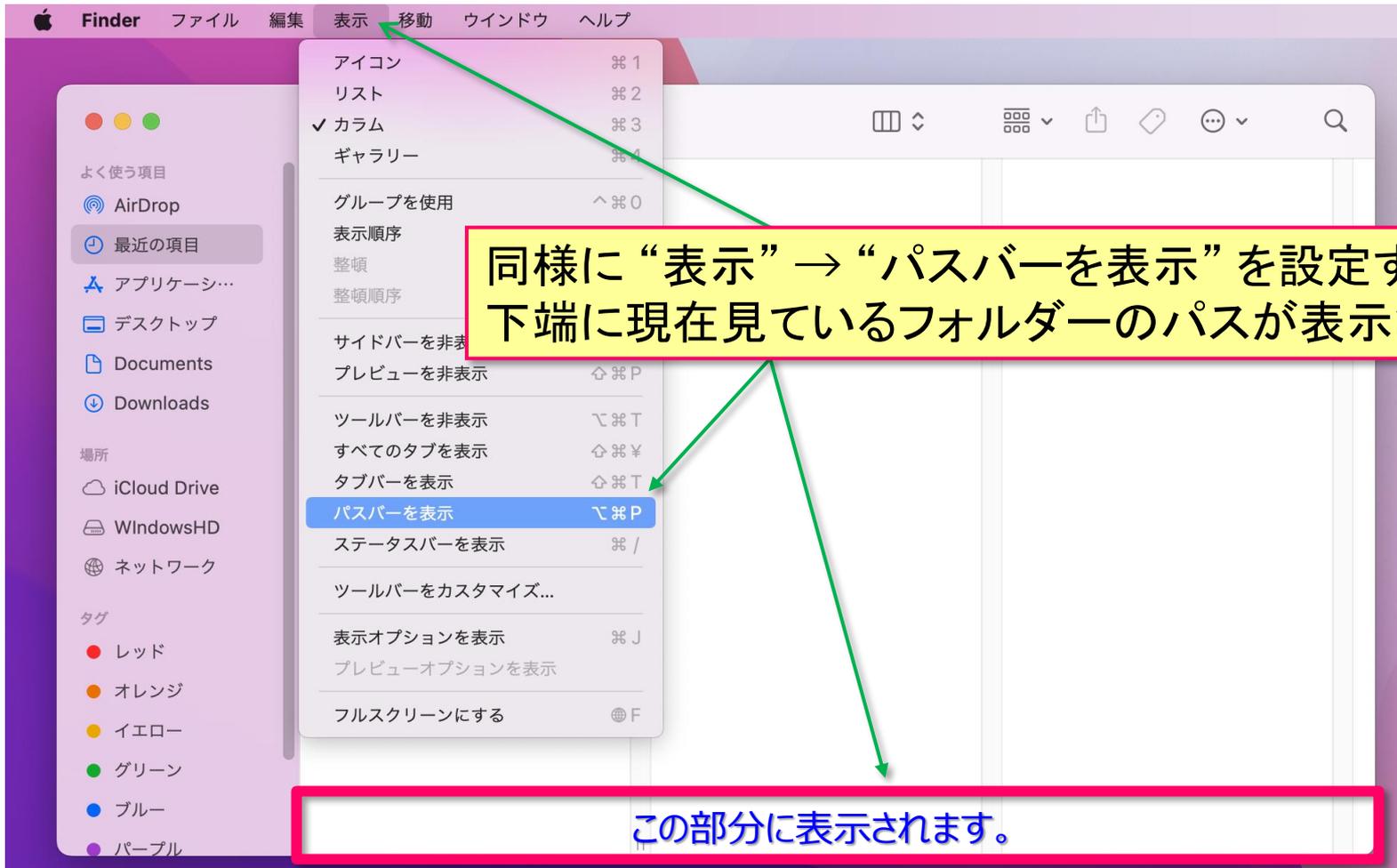
Dock 上にある Finder アイコンを  
右クリックして、  
“新規Finderウィンドウ” を選択

# 続き: Finder からファイル システムを触る (2)

Finder メニューから“表示”を選択すると、ファイル・フォルダの表示形式を変更できる。



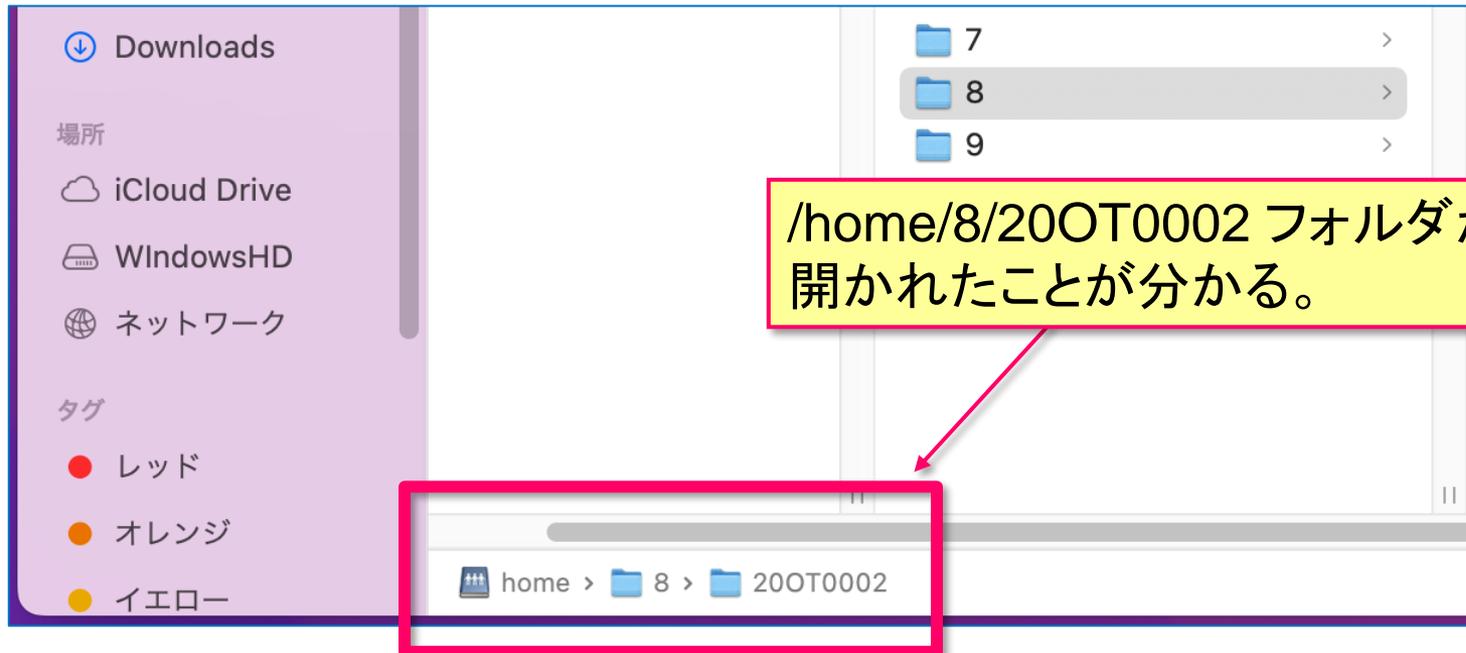
# Finder からファイル システムを触る (3)



# Finder からファイル システムを触る (4)



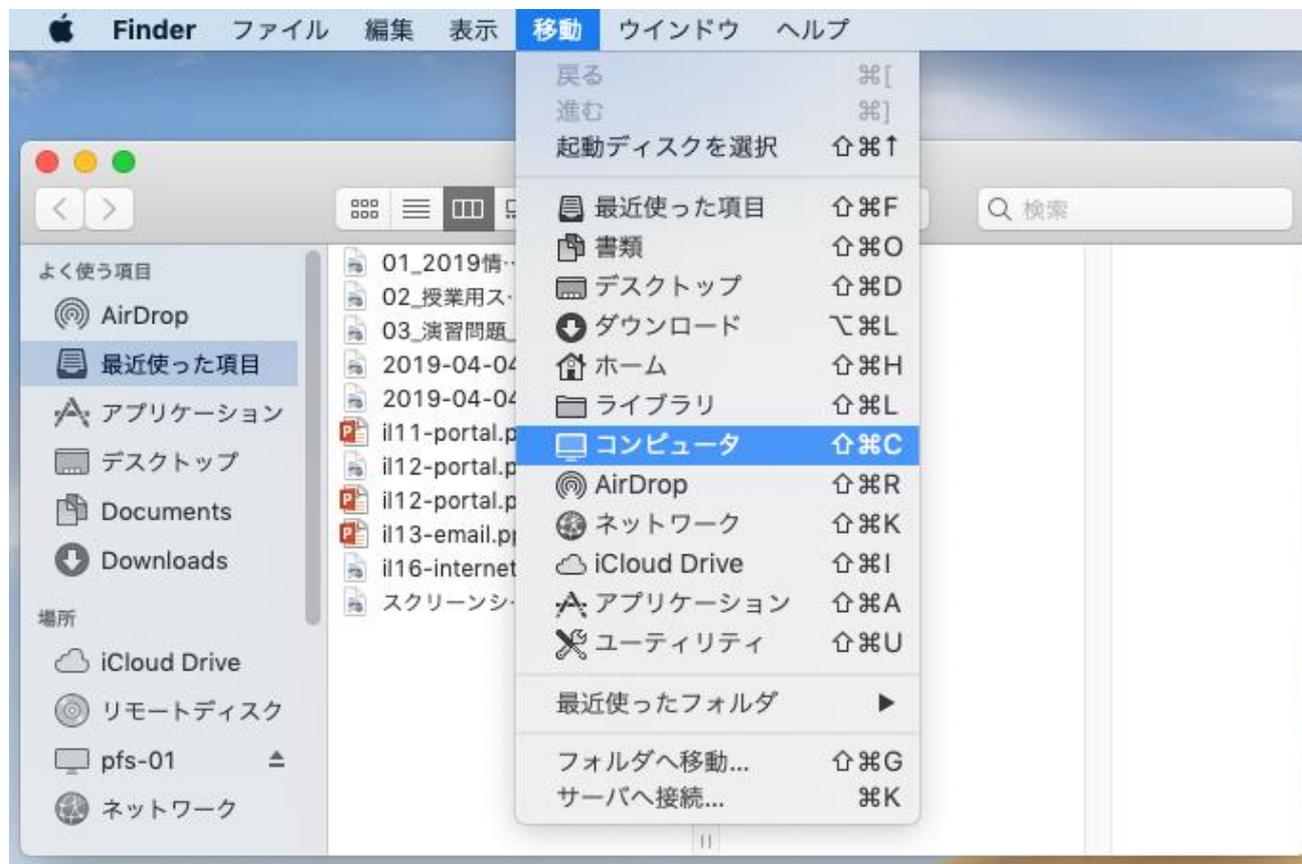
右端上にある自分のユーザ名が付いたアイコンをクリックすると…



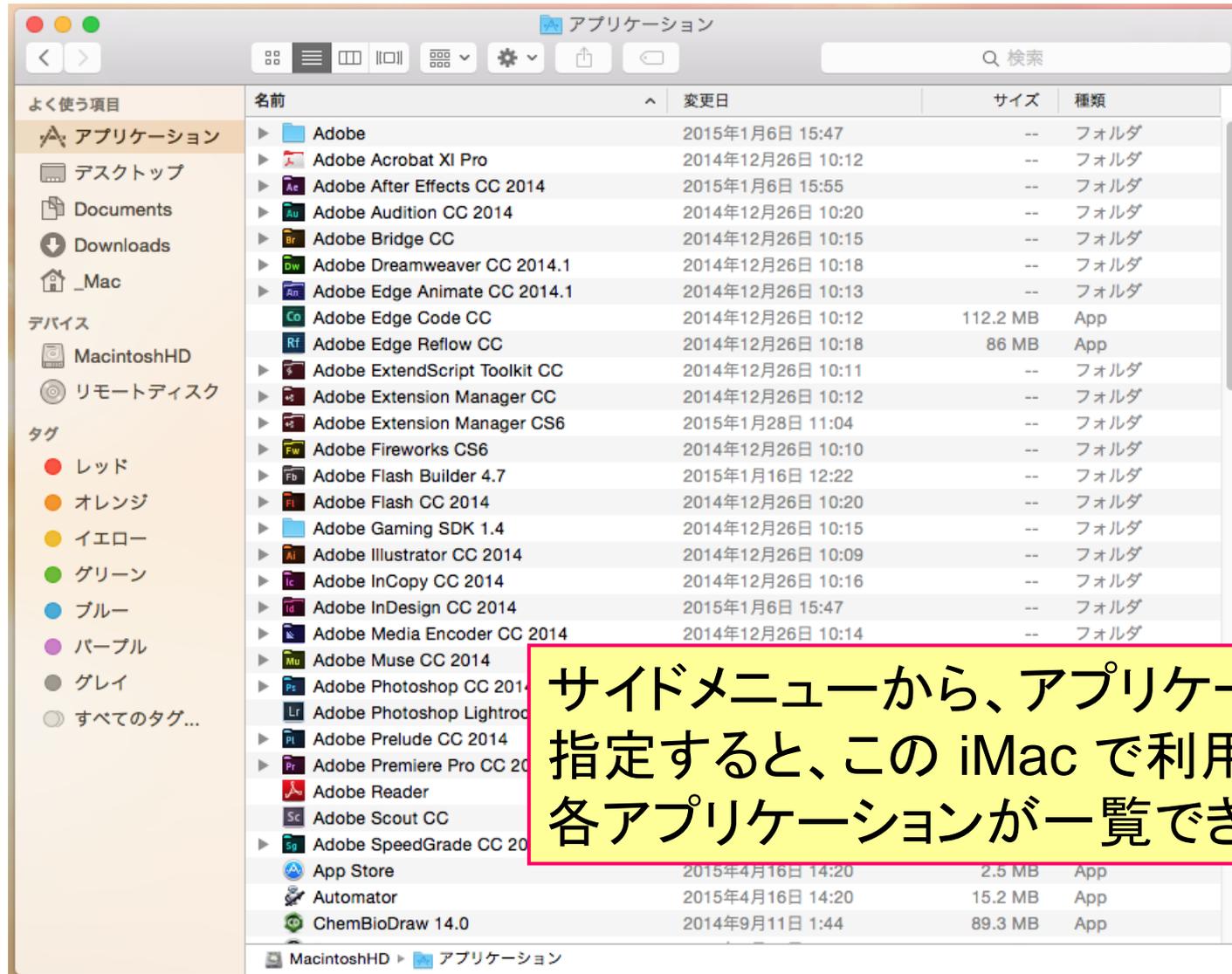
/home/8/200T0002 フォルダが開かれたことが分かる。

# Finder からファイル システムを触る (5)

Dock 経由の Finder メニューから“移動”をクリックし、表示されたリストから“コンピュータ”を指定する。利用している iMac の頂上フォルダに移動できる。



# Finder からファイル システムを触る (6)



# Dock の構成

本来の MacOS であれば、



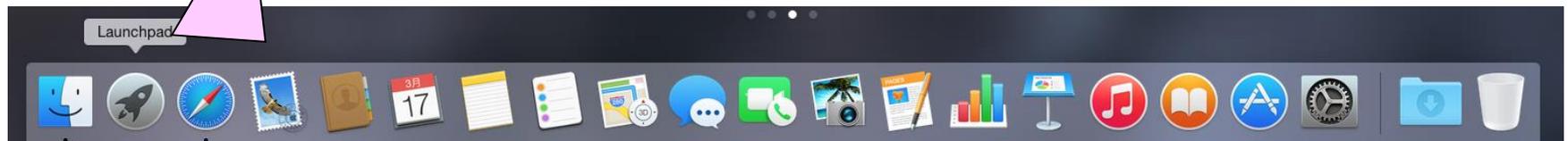
アプリとして登録できない場合は、ファイルとして登録しておける

# よく使うフォルダへ早くアクセスするには

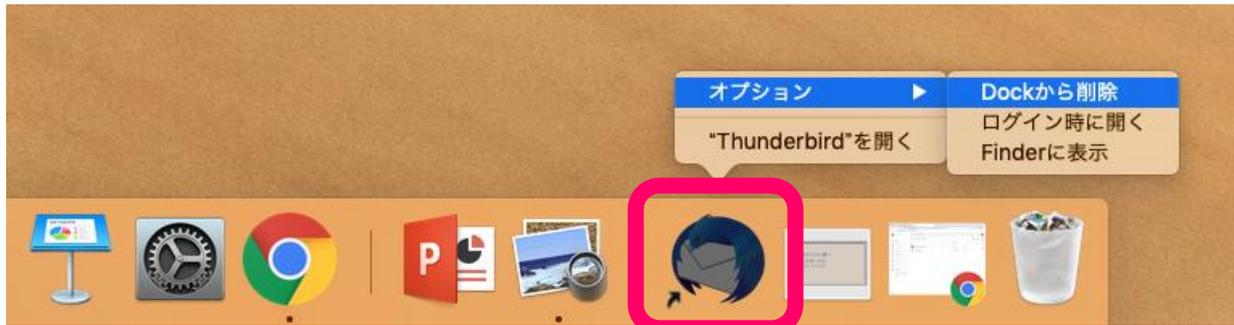
② ユーティリティを Dock まで Drag & Drop

- よく使うフォルダを Dock に登録しておくと、早くアクセスできるようになります。
- 例として、ユーティリティフォルダを登録してみましょう。

① ユーティリティが見つかるまでスクロール



# Dock からの削除



削除したいアイコンを  
右クリック

# コマンド ラインの利用

# コマンドの直接入力による操作

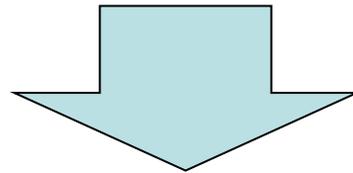
- GUI による操作は、直観的ではあるがシステム側の負担も増える。
  - GUI の制約や干渉, お節介機能による副作用
  - 処理負荷の増加による実行速度の低下/バグ発生
    - GUI 関連の部品は、作り直すというより増築という方針で開発される傾向にある ⇒ 動作が重くなる一方
    - 実行速度が低下すると必要なタイミングで処理できない場合が増える ⇒ 例外処理を完備するのは難しい
- 状況によっては、GUI を用いずにコマンドを直接入力した方が良い場合もある。
  - コマンドによる操作を覚えよう。

大前提：新しい発見をするために、様々な機能を自由に組み合わせたい。

# アーキテクトの真髓とは

---

- コマンド： 計算機における処理の単位だと考えると・・・
- どのくらいの粒度で処理単位を決めるか？
  - 処理単位を大きくすれば、**効率は良くなるが使い勝手は悪くなる。**
- どのくらい細かい処理を許容するか？
  - 細かい処理に対応すれば、**品質の高いサービスを提供できるが効率は悪くなる。**



- **アーキテクトの腕の見せ所**
- サービス品質／効率／規模／コストなどを**線形関係**に収めることができればベスト

# ターミナルの使い方（1）

---

- Mac では、“ターミナル” を起動してみましょう。
  - [アプリケーション] → [ユーティリティ] にあります。
  - Finder を起動すると、複数のファイルやフォルダを同時に表示しますが、ターミナルは何も表示しません。
  - ユーザが何かを入力すると、ターミナルはその結果を表示します。
- Windows では、“コマンド プロンプト” です。
  - エクスプローラと同様、左下の“スタート” 以下より、“Windows システム ツール” 内にあります。
  - もっと良いものもあるのですが、取り敢えず現在はこれを使います。

# ターミナルの使い方 (2)

---

- Finder やエクスプローラは、マウスにより現在注目しているフォルダを指定できます。
  - ファイル システムの階層も同時に見えるので、迷子になることはほとんどありません。
  - しかし、ターミナルにはウィンドウが一つしかありません。。
- **カレント フォルダ (カレント ディレクトリ)**
  - ターミナルには、**ユーザからのコマンド入力が、どのフォルダのファイルを対象にしているか**を管理するカレント フォルダ (カレント ディレクトリ) という内部データがあります。
  - **カレント フォルダを基準とした位置関係をイメージできるようにしよう (+ ファイルの場所を指定できるようにしよう)。**

# 代表的なコマンドの種類

---

- pwd: カレントフォルダの確認
- ls: フォルダ内のファイル名一覧を表示
- cd: カレントフォルダを移動
- less: ファイルの中身を表示
- mv: ファイルの移動
- cp: ファイルのコピー
- rm: ファイルの削除
- mkdir: フォルダ(ディレクトリ)の作成
- chmod: ファイルの許可情報を設定
- man: コマンドの使い方を表示

詳細は、教材にある  
リンクなどを参照

# コマンド入力のお助け機能

---

- ターミナルでは、コマンド入力を助けるため、以下の便利機能が用意されている。
  - C-p: 以前の入力を表示 (C-n: C-p の逆)
  - C-a: 入力の先頭へ移動 (C-e: 最後尾へ)
  - C-b: 1文字左へ移動 (C-f: 1文字右へ移動)
  - C-k: 最後尾まで削除 (C-y: 削除文字列を表示)
  - Return キーで確定
- コマンドに渡す文字列 ⇒ “引数” と呼ぶ

# コマンドの使用（1）

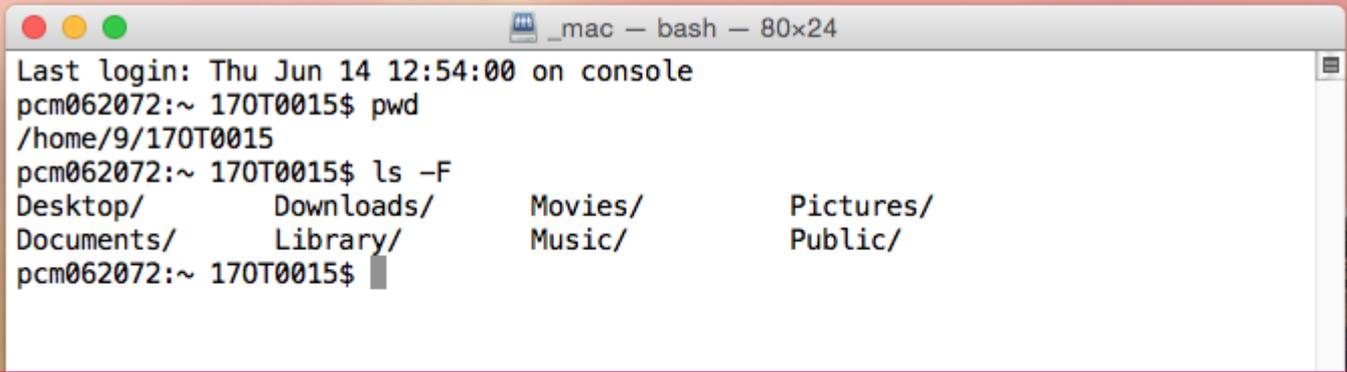
“pwd” コマンドで、カレントフォルダの位置を確認してみよう。  
表示された結果の意味を読み取れますか？



```
_mac - bash - 80x24
Last login: Thu Jun 14 12:54:00 on console
pcm062072:~ 170T0015$ pwd
/home/9/170T0015
pcm062072:~ 170T0015$
```

# コマンドの使用 (2)

“ls” コマンドで、カレント フォルダの中身を確認してみよう。  
ここで見えるものは、Finder から見た \_mac の中身と同じです。

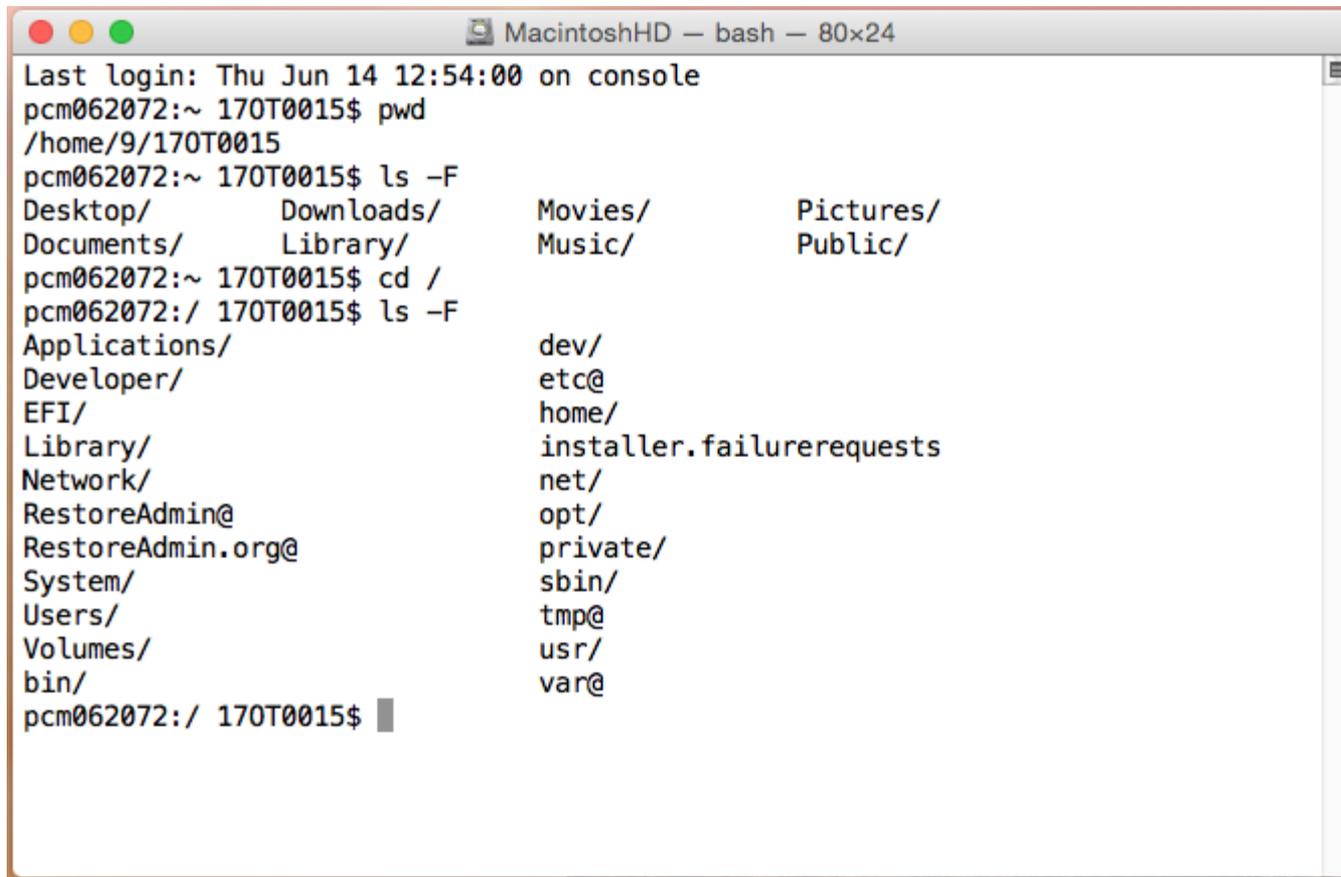


```
_mac - bash - 80x24
Last login: Thu Jun 14 12:54:00 on console
pcm062072:~ 170T0015$ pwd
/home/9/170T0015
pcm062072:~ 170T0015$ ls -F
Desktop/      Downloads/   Movies/      Pictures/
Documents/    Library/    Music/       Public/
pcm062072:~ 170T0015$
```

“ls” コマンドに“-F” オプションを付けると、  
フォルダとファイルの区別が付くように表示されます。

# コマンドの使用 (3)

“cd” コマンドでルートフォルダへ移動し、中身を確認してみよう。  
Finder から見たルートフォルダの中身と比べてみると・・・  
コマンド入力のお助け機能 (C-p など) を有効に使いましょう。



```
MacintoshHD -- bash -- 80x24
Last login: Thu Jun 14 12:54:00 on console
pcm062072:~ 170T0015$ pwd
/home/9/170T0015
pcm062072:~ 170T0015$ ls -F
Desktop/          Downloads/        Movies/          Pictures/
Documents/        Library/          Music/           Public/
pcm062072:~ 170T0015$ cd /
pcm062072:/ 170T0015$ ls -F
Applications/    dev/
Developer/       etc@
EFI/              home/
Library/          installer.failurerequests
Network/          net/
RestoreAdmin@    opt/
RestoreAdmin.org@ private/
System/           sbin/
Users/            tmp@
Volumes/          usr/
bin/              var@
pcm062072:/ 170T0015$
```

# コマンドの使用（4）

現在のカレント フォルダは、ルート フォルダになっています。ここから、自分のホーム フォルダへのパス名を指定して、“ls” コマンドで中身を確認してみましょう。

```
MacintoshHD -- bash -- 80x24
Last login: Thu Jun 14 12:54:00 on console
pcm062072:~ 170T0015$ pwd
/home/9/170T0015
pcm062072:~ 170T0015$ ls -F
Desktop/      Downloads/   Movies/      Pictures/
Documents/    Library/    Music/       Public/
pcm062072:~ 170T0015$ cd /
pcm062072:/ 170T0015$ ls -F
```

ホーム フォルダへのパス名は、ターミナルの起動直後に実行した“pwd” コマンドで表示されています。ルート フォルダから見て、ホーム フォルダは下にあるので、パス名は各経由フォルダを“/” で連結した形式になります。

```
bin/          var@
pcm062072:/ 170T0015$ ls -F home/9/170T0015
Desktop/      Downloads/   Movies/      Pictures/
Documents/    Library/    Music/       Public/
pcm062072:/ 170T0015$
```

# その他

コマンドラインを  
より深く利用するための機能

# ファイル名のパターンマッチ(1)

---

- **複数**のファイルを**同時に指定**したい。
- ファイル名の指定に？を混ぜる。
  - **？**の部分には、**どんな1文字**が入ってもOK
  - **a?b** で表わされるファイル名は、最初が a で最後が b の**合計3文字**であれば何でもよい。  
ex) aZb, a5b, ajb など
- 同、\* の扱い
  - **？**は1文字分だけだが、**\*は何文字でもOK**
  - **a\*b** で表わされるファイル名は、最初が a で最後が b であれば何文字でもよい。  
ex) aZb, a54321b, ajLkh66b など

# ファイル名のパターンマッチ(2)

---

- 同、[]の扱い

- **a[A - Z]b** で表わされるファイル名は、**最初が a** で **最後が b** となり、**その間**に含まれる文字は **A~Z** の **どれか**でなければならない。

- ex) aZb, aAKSSb など (ajLkh66b はダメ)

- このような表現形式をワイルドカードと呼びます。

- これと似た表現形式として正規表現があります。

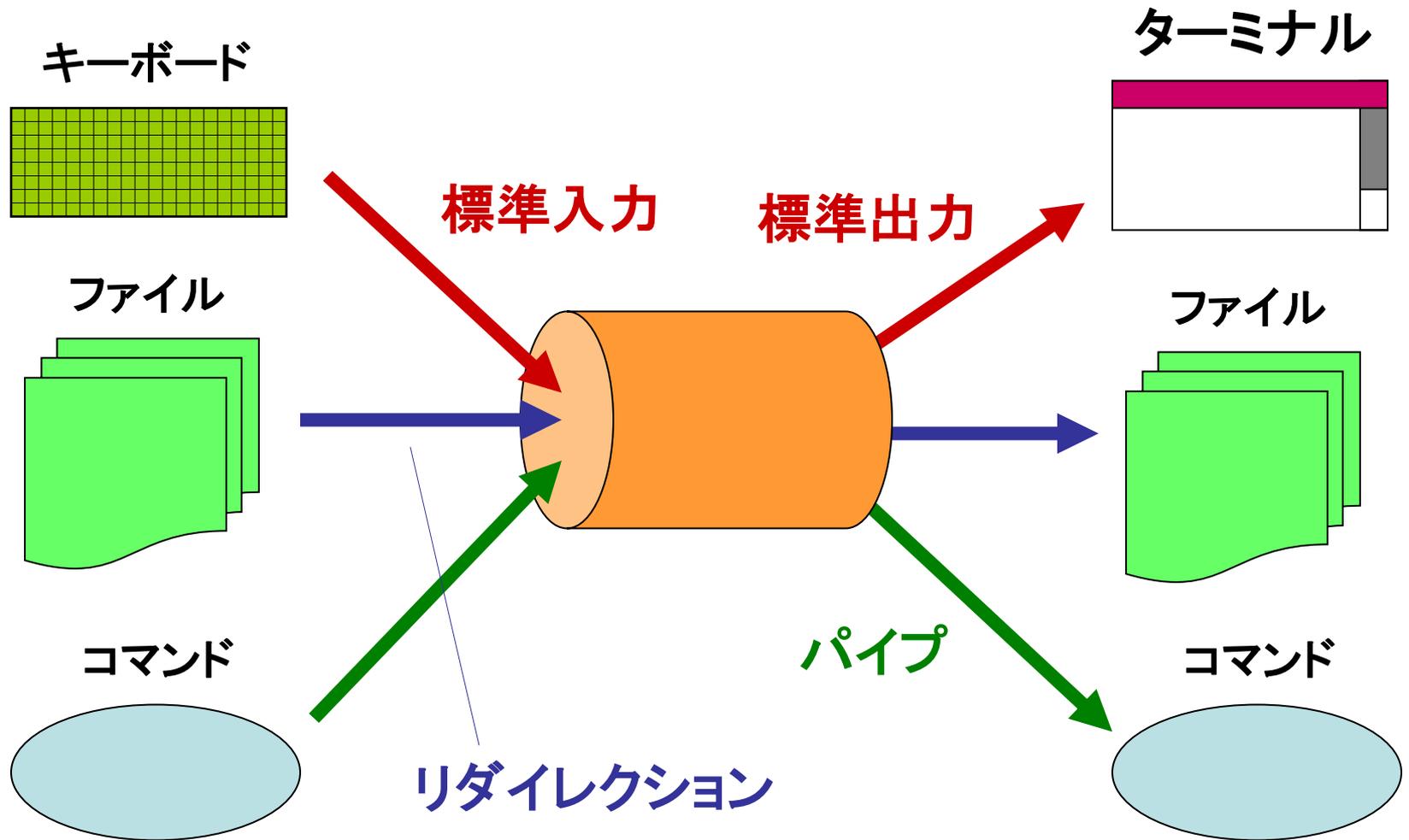
- コマンドの種類により、? や \* などを用いた表現形式でも意味が異なるので、注意が必要です。

# シェルによるコマンド処理

---

- コマンドの強制終了: C-c
- コマンドの一時停止: C-z
- バックグラウンド処理: 起動時 ⇒ &  
一時停止中 ⇒ bg
  - BG 処理 ⇒ 端末を占有しないでコマンドを実行する
- BG処理の一覧: jobs
- BG処理の終了: kill %job 番号

# ジョブの入出力



# リダイレクション

---

- ファイルへ出力: >
  - date > yyy.txt
  - date コマンドの結果を yyy.txt へ保存
- ファイルから入力: <
  - cat < yyy.txt
  - yyy.txt の内容を cat コマンドで表示
    - cat → 標準入力からの入力を標準出力に出力

# パイプ

---

- ジョブの出力を別のジョブの入力へ： |
  - date | wc
  - date コマンドの出力を wc コマンドの入力とする
    - wc → 入力を数えて、行数/単語数/文字数の順で表示

# エディタによる編集

# エディタの利用

## • 文書作成ツール

### – ワードプロ:

例えば Word では、フィールドコードと呼ばれる体裁処理コマンド (マークアップ) が編集する度に自動生成+埋め込まれている。

- 基本的には画面に表示されている通りに印刷される。
  - WYSWYG: What You See is What You get
- 書体の変更や文字以外のデータ取り込みなど機能が豊富
- 重い内部処理 → 処理の洪水で遅い/バグが多い。

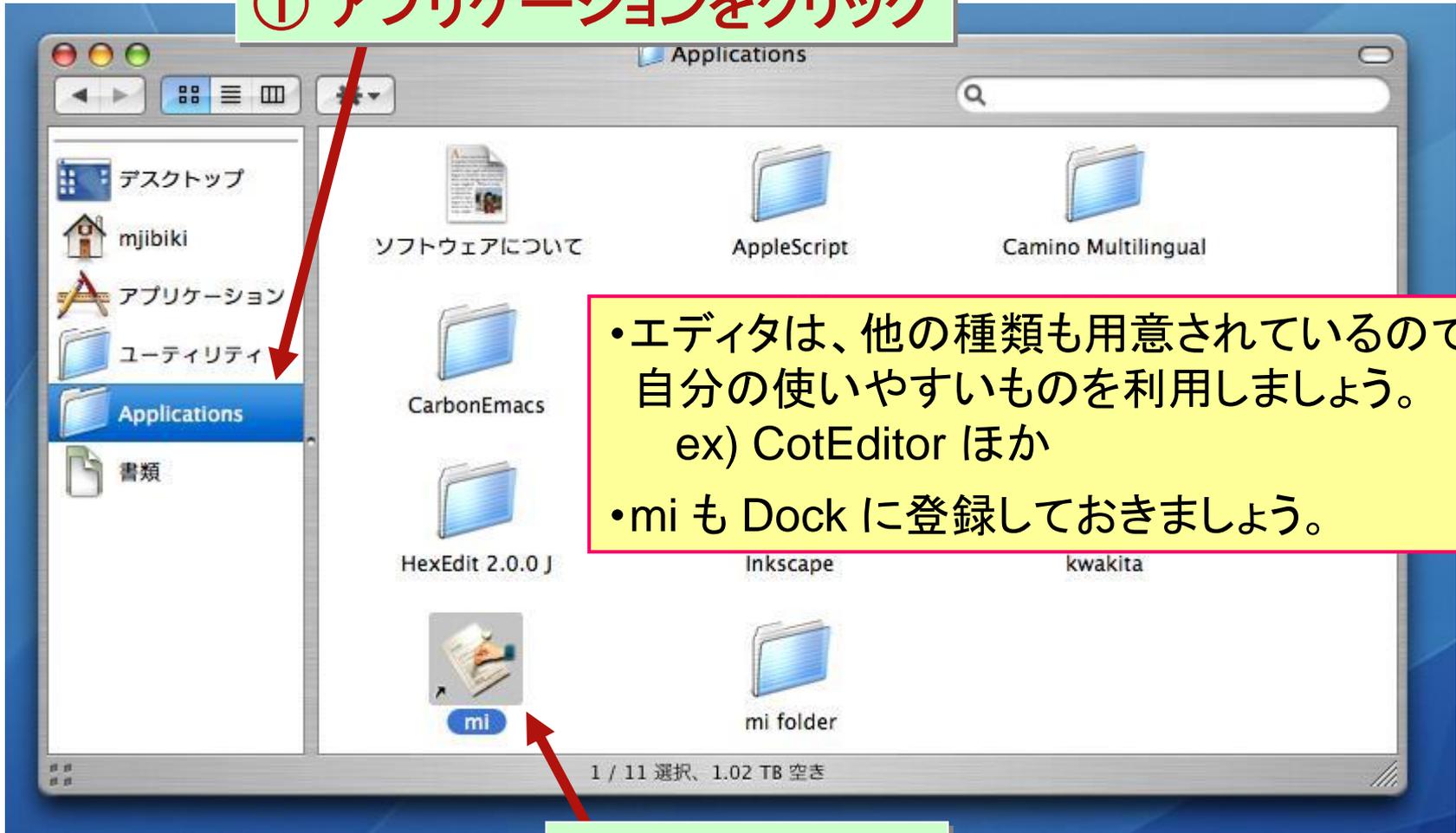
### – エディタ:

自動/手動の境界: システムの勘所

- 基本的には文字の入力のみ。
  - Windows のメモ帳はエディタの一つ。
- シンプルなので速い/バグが少ない ⇒ プロ用
  - 複雑な文書作成も**コマンドの埋め込みで可能 ⇒ マークアップ系**

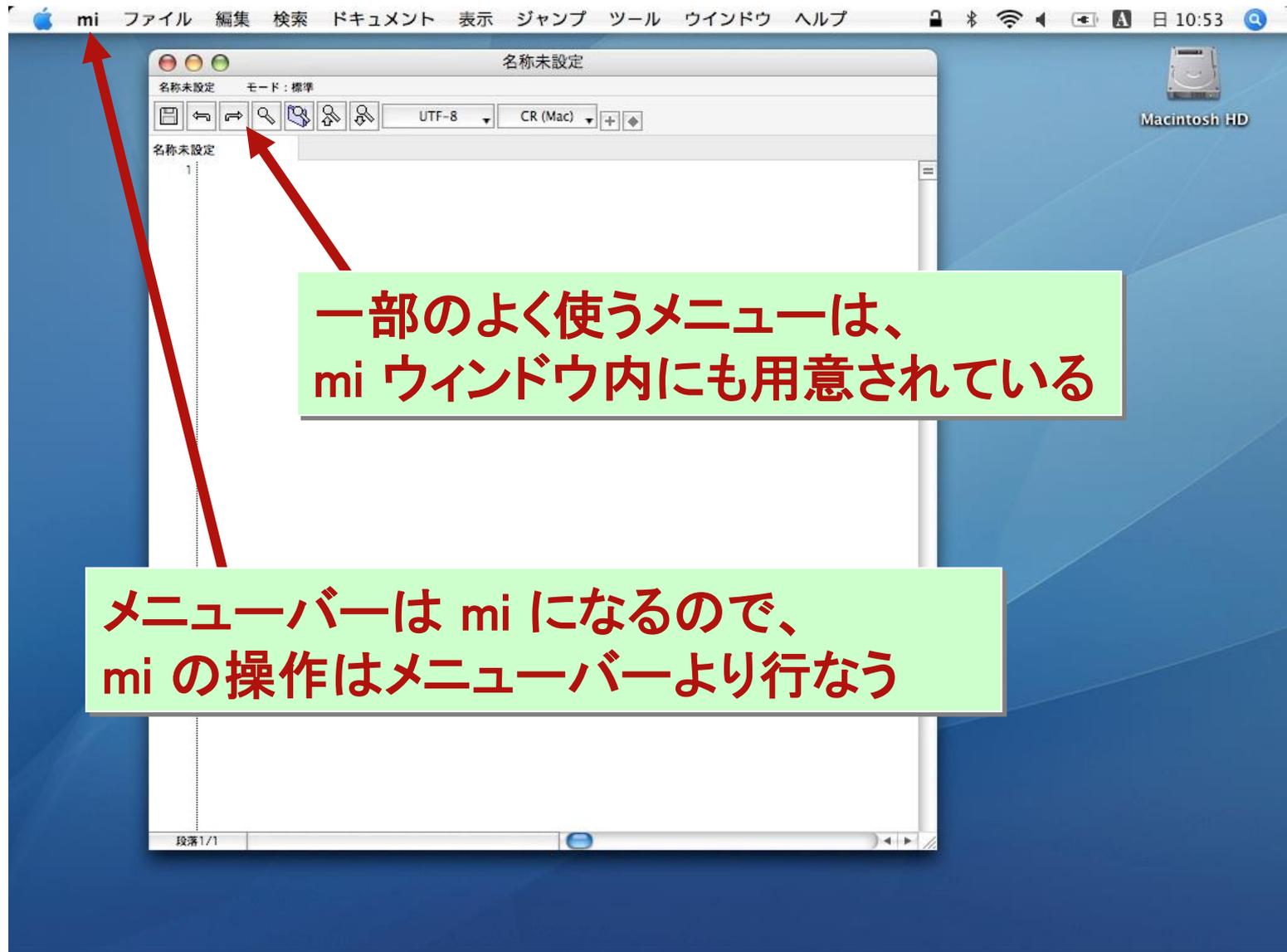
# mi の起動

① アプリケーションをクリック



② mi をクリック

# mi の使い方(1)

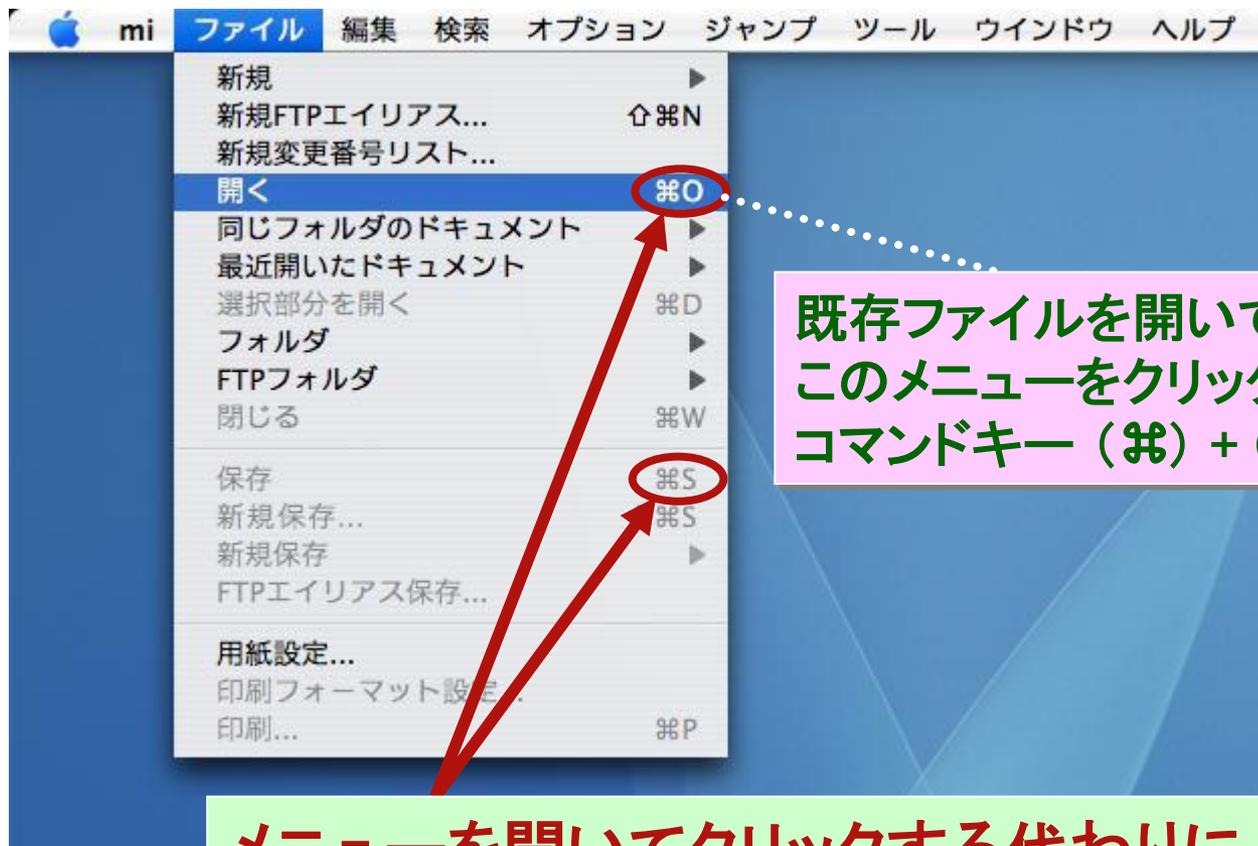


# mi の使い方(2)



mi で既存の (or 以前に保存した) ファイルを編集したい場合

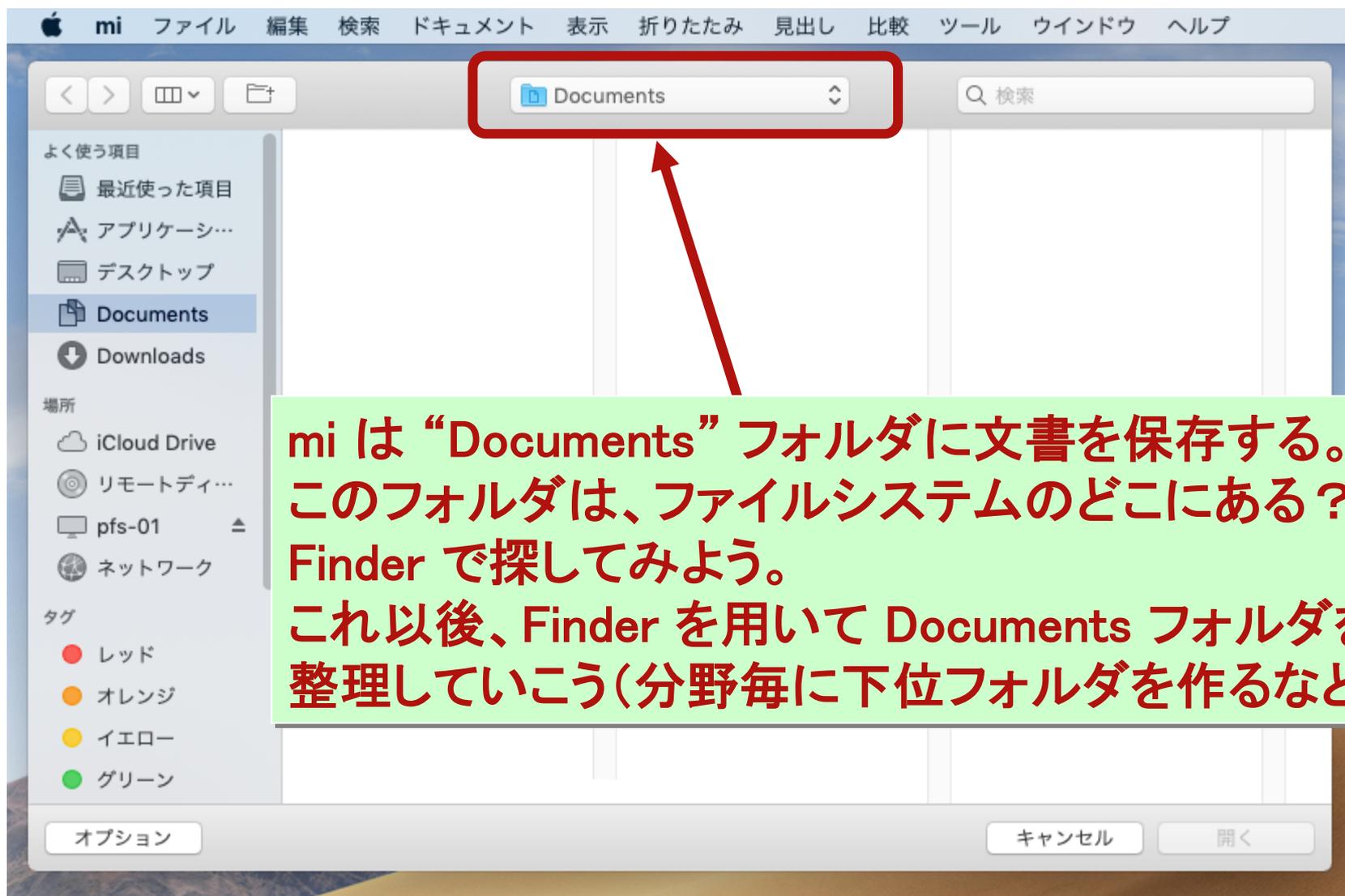
# mi の使い方 (3)



既存ファイルを開いて編集するには、このメニューをクリックする代わりに、コマンドキー (⌘) + O を押せばできる

メニューを開いてクリックする代わりに、対応するキーを押すことで機能を利用できる (慣れてくればこちらの方が速い)

# mi の使い方(4)



mi は“Documents”フォルダに文書を保存する。  
このフォルダは、ファイルシステムのどこにある？  
Finder で探してみよう。  
これ以後、Finder を用いて Documents フォルダを  
整理していこう(分野毎に下位フォルダを作るなど)。