

# ROOT 講習会 第1回 資料

Keita Mizukoshi (YMAP, JAXA)  
Mail: mizukoshi.keita@jaxa.jp

- 本資料の著作権、文責は著者に帰属し、所属機関を代表したり、機関の意見を表明するものではありません。
- 学校、研究機関の教育、研究目的であれば自由に使用することができます。報告なく改変、再配布可能です。
- ただし使用者は誤りや誤字を報告する義務があります。

# 自己紹介

- 水越 豪太
- JAXA ISAS大気球グループ
- GAPS実験
- その前は地下実験  
(XENONnT, CANDLES)
- 先生ではないので気軽になんでも  
聞いてください
- どこかで見かけたら声をかけて  
ください(飲みにいきましょう)



# 前置き

- あなたがわからないことは他の人もわかっていない
  - 私の説明が悪い
- 代表して質問してください
- この時間を'あなたにとって'有意義な時間にしてください.
- 私の説明を止めて口頭で質問してください
- チャット/Slackでもよいです
- Topicが簡単すぎる時はスキップご自由に
- 2019年度までの講師 奥村さんの資料も有用です.
  - <https://github.com/akira-okumura/RHEA>

# 講習会の到達目標

- 物理解析の雰囲気を把握する
  - 解析の基礎的な概念を理解する
  - ROOTで絵(ヒストグラム, グラフ)が描けるようになる
  - 自力で必要な機能の用法をWebで検索できる
  - ROOTを題材にして, 環境やツールの変化で枯れない最低限の知識を習得する.
- 
- 実際には研究ですぐにROOTを使うと思うので,  
必要に迫られつつROOTをやっていきましょう

# プログラミング言語

- Fortran
  - 数値計算用に開発 非常に高速
  - 大規模な開発は難しい
  - 開発資源が貧弱だった際の名残が多い
- C++
  - 処理速度が高速
  - 大規模開発を言語仕様で想定
  - 各種デバッグツールが揃っている
  - なんでもできる
- python
  - 簡単(とよく言われる)
  - 普通に使うと低速
  - 機械学習だとよく使われる
- shell script
  - コマンドライン向けのスクリプト言語
- Java
  - 高速, ライブラリ充実, 大規模開発可能
  - 商用でよく使われる(数値計算での使用はごく稀)
- Javascript
  - Webでよく使われる
- SQL
  - データベース操作用

フルチューンすれば非常に高速  
初心者は 計算時間 << 開発時間 なのでまずは開発が比較的易しい言語で始めるのを勧める  
C++コンパイラが賢いので優位性は埋まりつつある

計算機のハードウェアを扱うのも得意 ~ 高速  
現在も進化を続けている言語  
C++に慣れれば他の言語をつかうのも簡単になる

ライブラリが充実していて最近流行の言語  
スクリプト言語なので速度はそれなりだが,ライブラリはC++で書かれていることが多く,一部高速  
C++とあわせて身に付けることを勧める

他にも色々言語はあるが,数値計算は守備範囲外  
目的にあった言語を選択すると,開発が容易になる

# 解析環境

- GNU plot ← command line
  - グラフを書くためのソフトウェア
  - お絵かきが上手
  - 絵を描くまでの複雑な計算はサポートしていない
- paw / anapaw ← Fortran
  - Fortranベースの解析ツール
  - ROOTが出てくる前に使われていた
- **ROOT ← C++**
  - C++ベースの解析ツール
  - 非常に高速
  - 大規模なライブラリ
  - 重いデータを簡単に扱える
  - pyROOTというpython経由で使えるシステムあり
- numpy+matplotlib+etc. ← python
  - pythonだけで完結する環境
  - numpyやscipyといった数値計算用のライブラリとmatplotlibという絵を描く用のライブラリを組み合わせて使う
  - 賢く使わないと非常に低速

ROOTをいれるまでもない環境/解析で稀に使う  
解析部分は貧弱なので,結局ROOTを使うことも

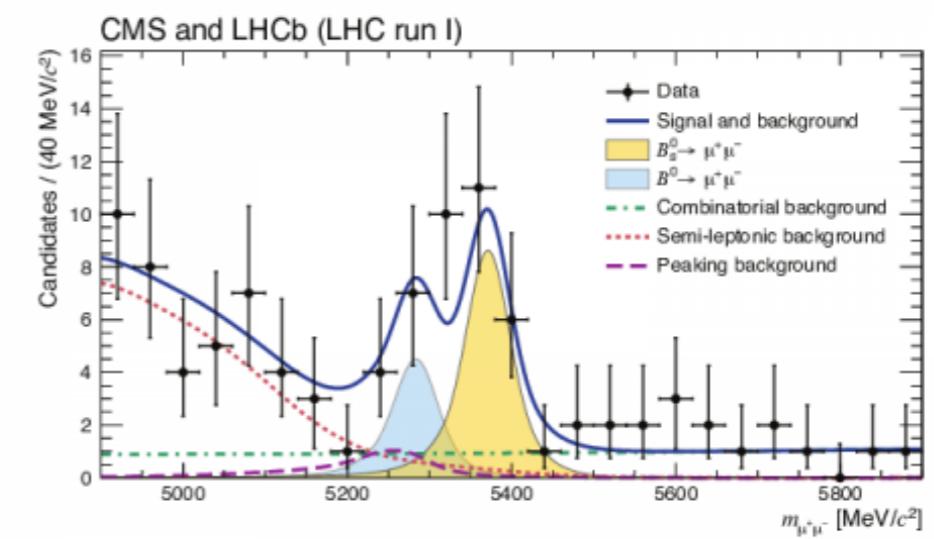
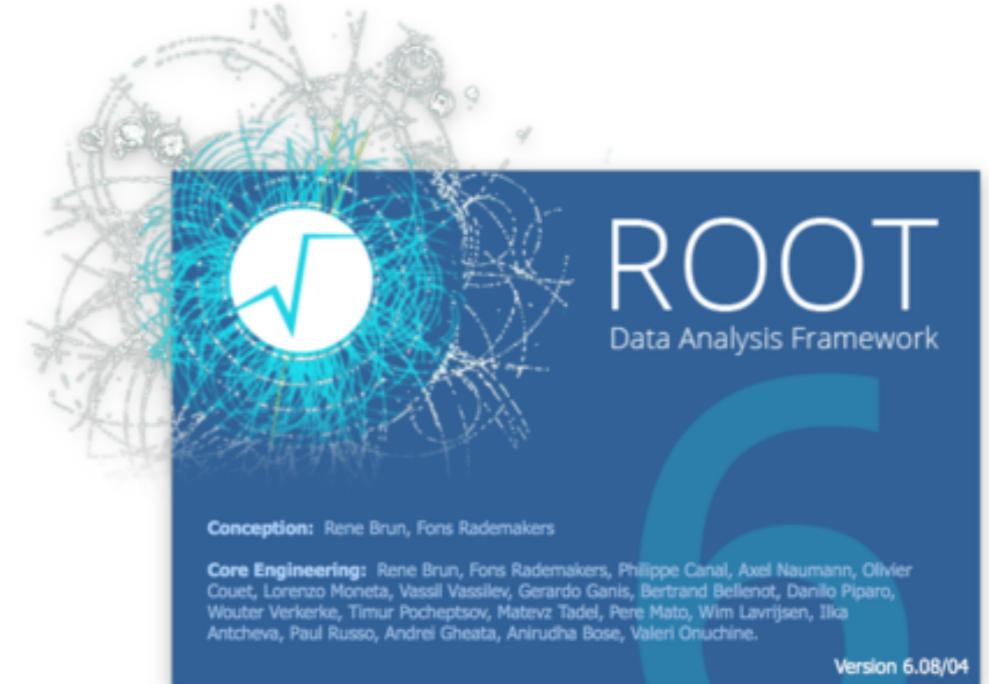
いまから始める人はわすれてよい  
指導教員がpawを使えと言ってきたら,いい機会  
なのでROOTでかきなおしてあげましょう

本講習会の主題  
ROOTさえ入れてしまえば,データを保存し,  
解析して,絵を描くということが全てできる  
物理解析に必要なものが全て揃っているので,  
業界標準で使われている

最近のpythonの流行もあって,pythonベースで  
解析している実験(i.e., XENON, IceCube)もある  
しかし,ROOTが便利(特にtree, histogram)  
なので結局根元はROOTにしばしば依存  
絵はROOTよりも綺麗

# ROOT とは

- 世界中で科学計算,データ解析のために用いられているツール
  - 莫大なデータを扱う際 (~ほぼ全ての物理実験) の事実上の標準
    - 最近はpythonも頑張っているが...
    - データを可視化したり複雑な数値計算を高速に実行可能
    - 独自記法は少なく C++ の知識で動かすことができる.



<https://root.cern.ch>

# わからないことがあったら... (質問のしかた)

- コードが思うように動かない時,
- まずエラーメッセージやログ出力をちゃんと読む(英語でも諦めない)
  - 全く意味がわからない時はGoogle翻訳へ  
(ただし,現段階ではGoogle翻訳をしても意味がわからないことが多い)
- 出力をコピー&ペーストしてGoogle検索
  - わからないことは大抵先人も分からなくて質問している
  - ただし,自分の環境に依存する部分(自分のユーザ名など)は外して検索する
- 先輩/先生に質問する
  - エラーメッセージを一字一句変えずに送る
  - スクショや写真よりメッセージをコピペで送る方がよい
  - エラーを解決するために試したことば全て伝える
- 頼れる先輩/先生が近くにいない
  - YMAPに入会すれば頼れる先輩がたくさんできます(無料です!)
- ネットできく
  - Yahoo!知恵袋よりも,Software専門の所で質問する方が有益です.
    - ROOT 公式Forum, StackOverFlow
    - 複数の場所で聞くことをしない
    - 自己(や先生や先輩)解決した場合も結果を載せておく
- Original Idea: Akira Okumura ROOT講習会2019



# 質問の例

- Amazon Web serviceの "技術的なお問い合わせにかんするガイドライン" に良い質問と悪い質問の例がたくさん載っている
  - AWSに限定しない汎用の内容
  - <https://aws.amazon.com/jp/premiumsupport/tech-support-guidelines/>
- とはいえ、本講習会では適当でもよいので気軽に質問してください。
- 卒業時に体裁の整った質問ができるくらいが目標です。

## 例

- ✗ 「Error: Invalid parameters なんとか」って、どういう意味ですか？
- ✗ 「Error: Invalid parameters. Exceed capacity limit」は、どういう時に出ますか？
- \*\*\* を実現するために、CLIで「xcmd ??? -o -k -v:33」と投入したら、「Error: Invalid parameters. Exceeding capacity limit」とメッセージが出て、失敗しました。どうしたら良いでしょうか。コマンド投入時刻は日本時間の本日 15:52、Request ID は \*\*\* です。

## 例

- ✗ いろいろやっていますが、うまくいきません。
- 下記のURLの手順は試しましたが、最後のステップで\*\*\* というエラーがです。  
[https://docs.aws.amazon.com/\\*\\*\\*/\\*\\*/](https://docs.aws.amazon.com/***/**/)

## 例

- ✗ 東京リージョンの EC2 (i-\*\*) が、応答しません。
- 東京リージョンの EC2 (i-\*\*) が、オンプレからの ping に反応しません。

# 写経のススメ

- 本講習会で示すソースコードは全て公開されています。
- Slackでもソースコード例を適宜示します。
- しかし、コードをコピペせずに自身の手で入力すること(写経)を強くお勧めします。
  - 写経を通じて,C++/ROOTの構文を学ぶ
  - コードの構造やアルゴリズムを追跡する
  - タイピング速度の向上
- できたらコードの一部を変更してみる
- いったん写経習慣を身につけて、アホらしくてやってられなくなるまで写経しましょう
- 講習会では時間を長めにとります。

# シェル(Shell)

- ・ コマンドライン(ターミナル)をつかうモチベ説明
- ・ コマンドラインでコマンドを打ち込めるようになったときShellと呼ばれるプログラムが起動している
  - ・ コンピュータと人間がやり取りするためのインターフェース
- ・ 複雑で大規模な作業をしたいとき,大抵既製品はない
  - ・ 自分でつくってshellから起動する

# 例:写真の加工

- 写真に撮影時刻を(昔の現像写真のように)付記したい
- もちろん既成のアプリをつかってできる
  - 撮影日時は正しいのか?
  - 同じ位置に入れるには?
  - 日付のみ?時刻も?
  - 何十枚もあったら?
  - Shellで解決できる



# コマンドを使う

- シェルからコマンドをつかう
  - この場合はexiftoolコマンド
  - 写真のメタデータを見ることができる
- sedコマンドで不要な部分を切り出し
- convertコマンドで写真に文字を載せる
- shell scriptで数行で実現

```
> exiftool my_friend.jpg
ExifTool Version Number : 11.85
File Name               : my_friend.jpg
Directory              : .
File Size               : 2.3 MB
File Modification Date/Time : 2020:04:07 15:15:40+09:00
File Access Date/Time   : 2020:04:21 18:30:30+09:00
File Inode Change Date/Time : 2020:04:21 18:30:28+09:00
File Permissions        : rwxrwxrwx
File Type               : JPEG
File Type Extension    : jpg
...
ion to extract)
Red Balance             : 1.65625
Scale Factor To 35 mm Equivalent : 5.6
Shutter Speed           : 1/50
Create Date              : 2020:04:07 15:15:40+09:00
Date/Time Original       : 2020:04:07 15:15:40+09:00
Modify Date              : 2020:04:07 15:15:40+09:00
...
...
```



```
# shell_image_date.sh
1 #! /bin/sh
2
3 PIC="$1"
4 [ -z "$PIC" ] && exit
5
6 DATE=$(exiftool -DateTimeOriginal -d "%Y-%m-%d %H:%M" -S "$PIC" | sed 's/DateTimeOriginal: //')
7 NAME=$(basename $PIC) NAME="$NAME"
8 echo "new image = $NAME"
9 convert $PIC -fill yellow -pointsize 80 -undercolor '#00000080' -annotate +2700+2700 "$DATE" $NAME
```

# 繰り返し

```
> ./shell_image_date.sh my_friend_1.jpg  
> ./shell_image_date.sh my_friend_2.jpg  
> ./shell_image_date.sh my_friend_3.jpg
```

- スクリプトを作れば後は繰り返しだけ
  - この繰り返しも自動化できる  
(が,本講習の範囲外)
- Shellでの作業の汎用性
- マウスで作業するより早い
- 作業が多くなっても対応可能
- 小さなツールを組み合わせて仕事をする
  - 組み合わせで可能性は無限
- 単に例として写真加工を示した
- このスクリプトを理解する必要はない



# コマンドラインを起動

- ターミナルソフトを起動
  - Windows→WSL2, MobaXterm など
  - Mac→Terminal.app または iTerm など
- とにかく何か文字が打ち込める状態になったら  
キーボードからコマンドを入力してEnterキーを押してみる
  - 下の例だと先頭の'>'は打ち込まない文字です。(あなたの環境によって\$や%かも)
  - 単に'T's' 'Enter' と押してみてください
  - 各人の環境によって多少ターミナルの飾りは異なります.
  - できたらそのあと'mkdir root\_lecture'もやってみてください.
    - 作業用のディレクトリを作ります.

```
> cd
> ls
Applications/           Downloads/          xenon/
> mkdir root_lecture
> cd root_lecture
> pwd
/Users/mzks/root_lecture
```

# ターミナル/端末 の操作

- 色々と飾りが付いていても基本は同じ
- コマンドを打ち込むことで実行される
- ターミナルの見た目改善記事
  - <https://qiita.com/kinchiki/items/57e9391128d07819c321>
  - ここまで拘らなくても良いが、多少時間をかけると将来にわたって時間を節約できる

ターミナル/端末 というソフトウェアを起動

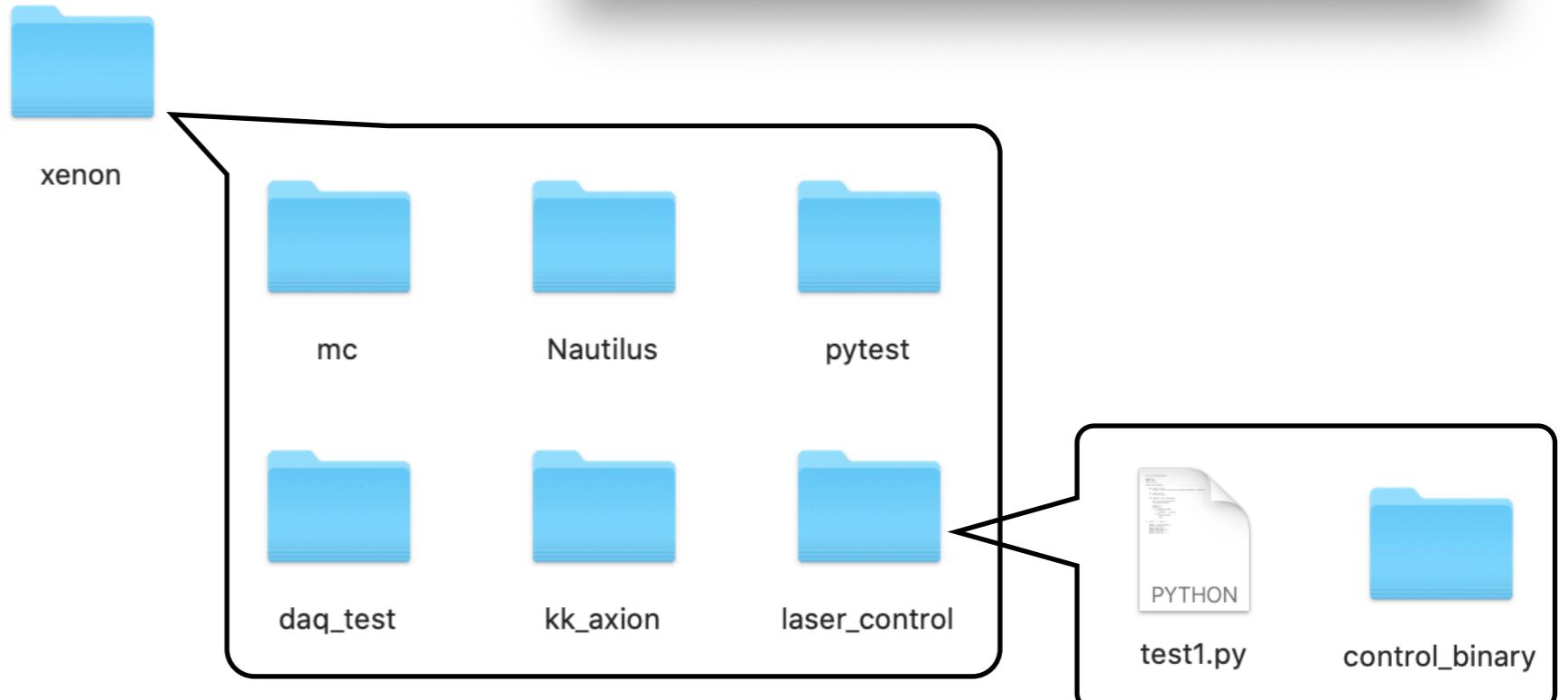
```
mzks — zsh — 67x10
>Last login: Tue Apr 21 21:47:26 on ttys007
[mzks@gold] ~
[% pwd
/Users/mzks
[mzks@gold] ~%
```

Prompt %, \$, > とか  
ここにコマンドを  
打ち込む

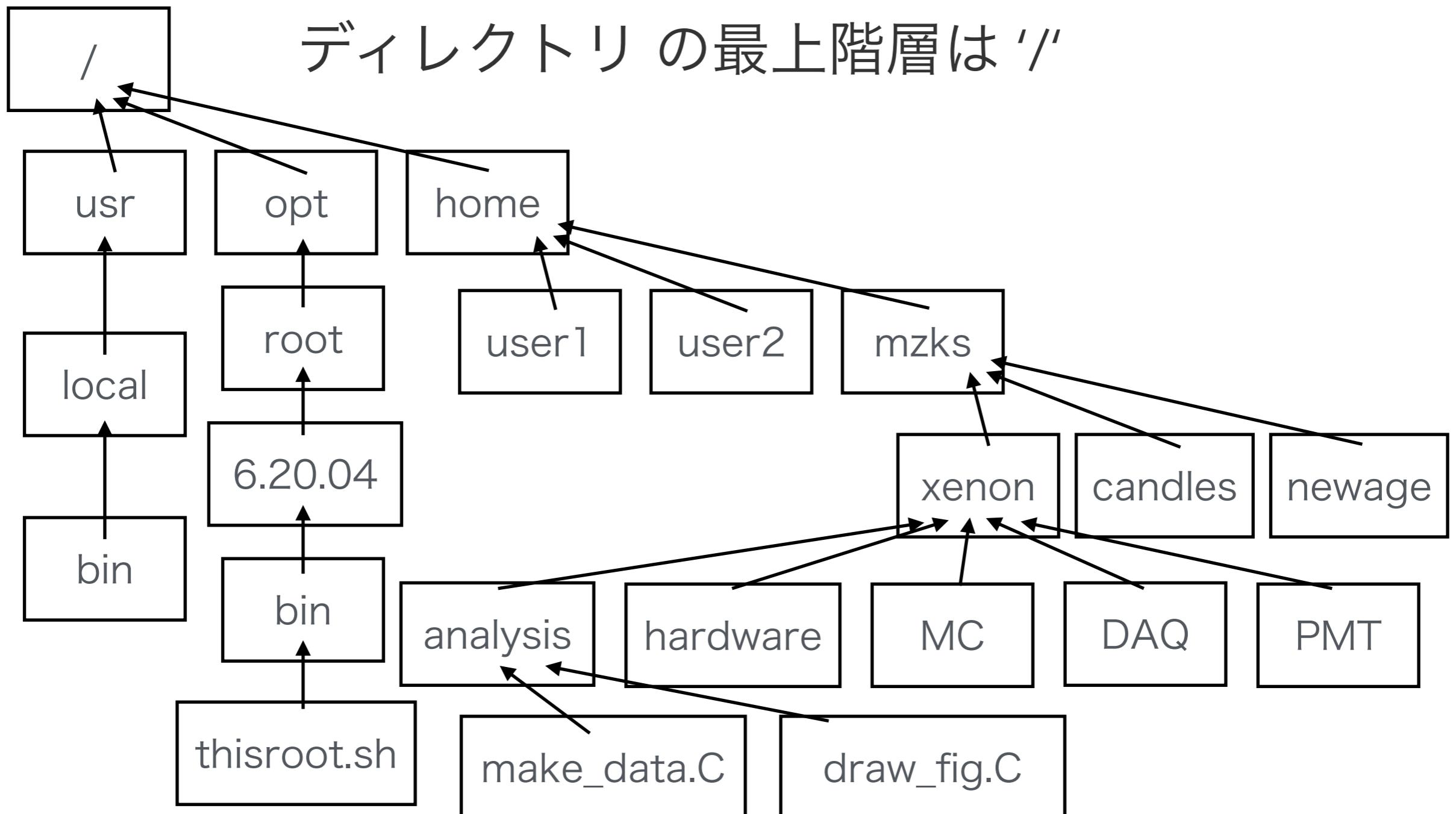
# ディレクトリ構造

- Unix/Linuxシステムでは、WindowsやMacのフォルダ、ディレクトリのように入れ子構造で管理する

Name	Date Modified	Size
Applications	August 14, 2019 14:19	
candles	March 4, 2020 14:22	
chisoko	April 7, 2020 15:35	
Creative Cloud Files	Yesterday 1:44	
Downloads	Today 6:31	
fonts	June 8, 2019 18:04	
gakushin	March 18, 2020 10:24	
intel	June 15, 2019 19:43	
Movies	December 2, 2019 11:18	
Music	December 2, 2019 11:18	
newage	April 15, 2020 19:57	
Parallels	June 9, 2019 22:46	
Pictures	December 23, 2019 15:45	
plots	June 18, 2019 14:10	
pp_a	March 24, 2020 16:22	
Projects	April 15, 2020 22:30	
Public	June 6, 2019 18:28	
root_lecture	April 21, 2020 18:58	
sunaba	December 21, 2019 19:01	
verapdf	January 9, 2020 17:22	
watch_pp	April 3, 2020 16:43	
WebstormProjects	April 13, 2020 3:56	
www	December 20, 2019 14:37	
xenon	April 9, 2020 20:17	

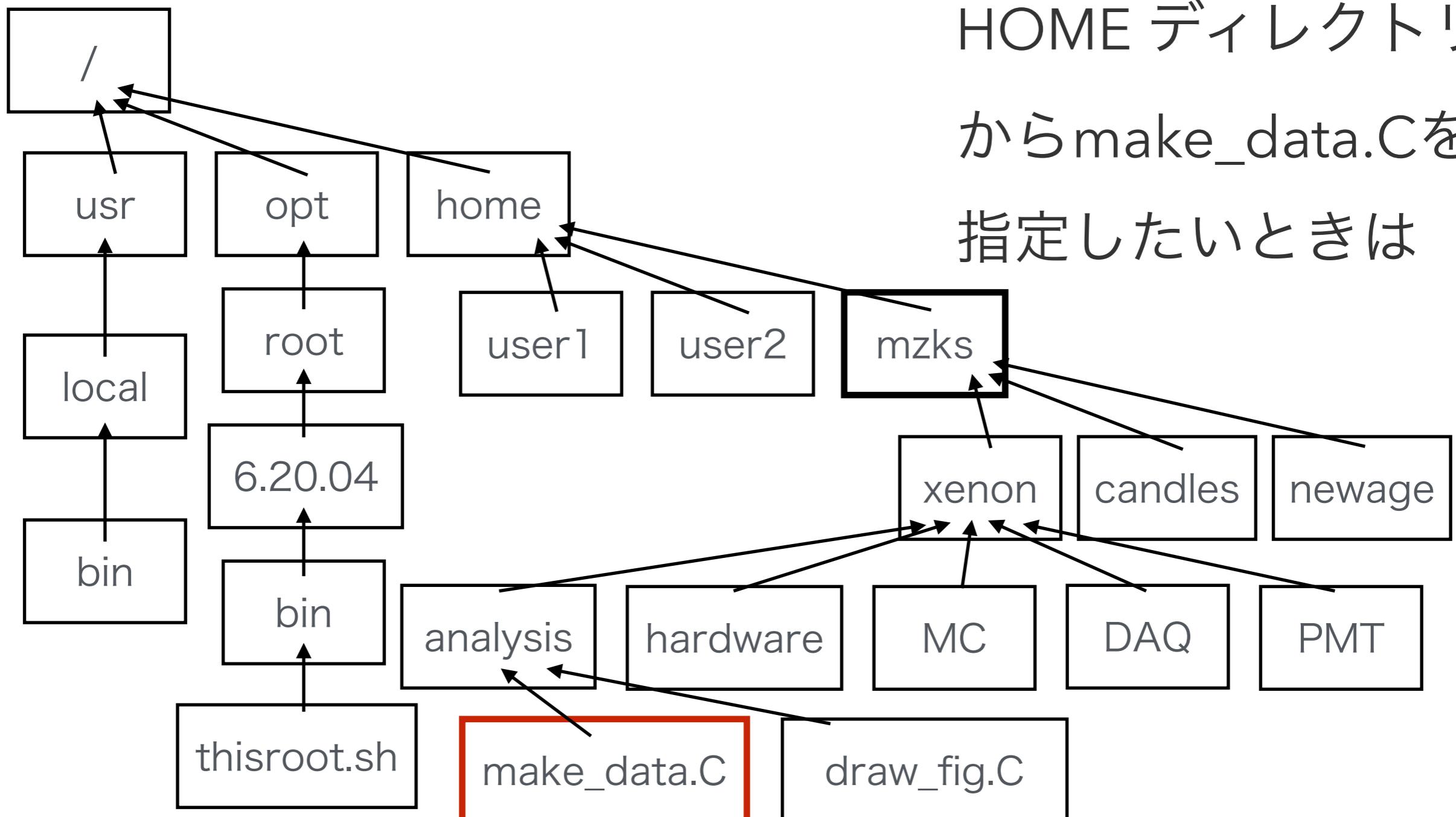


# Path構造



- 階層構造で管理されている

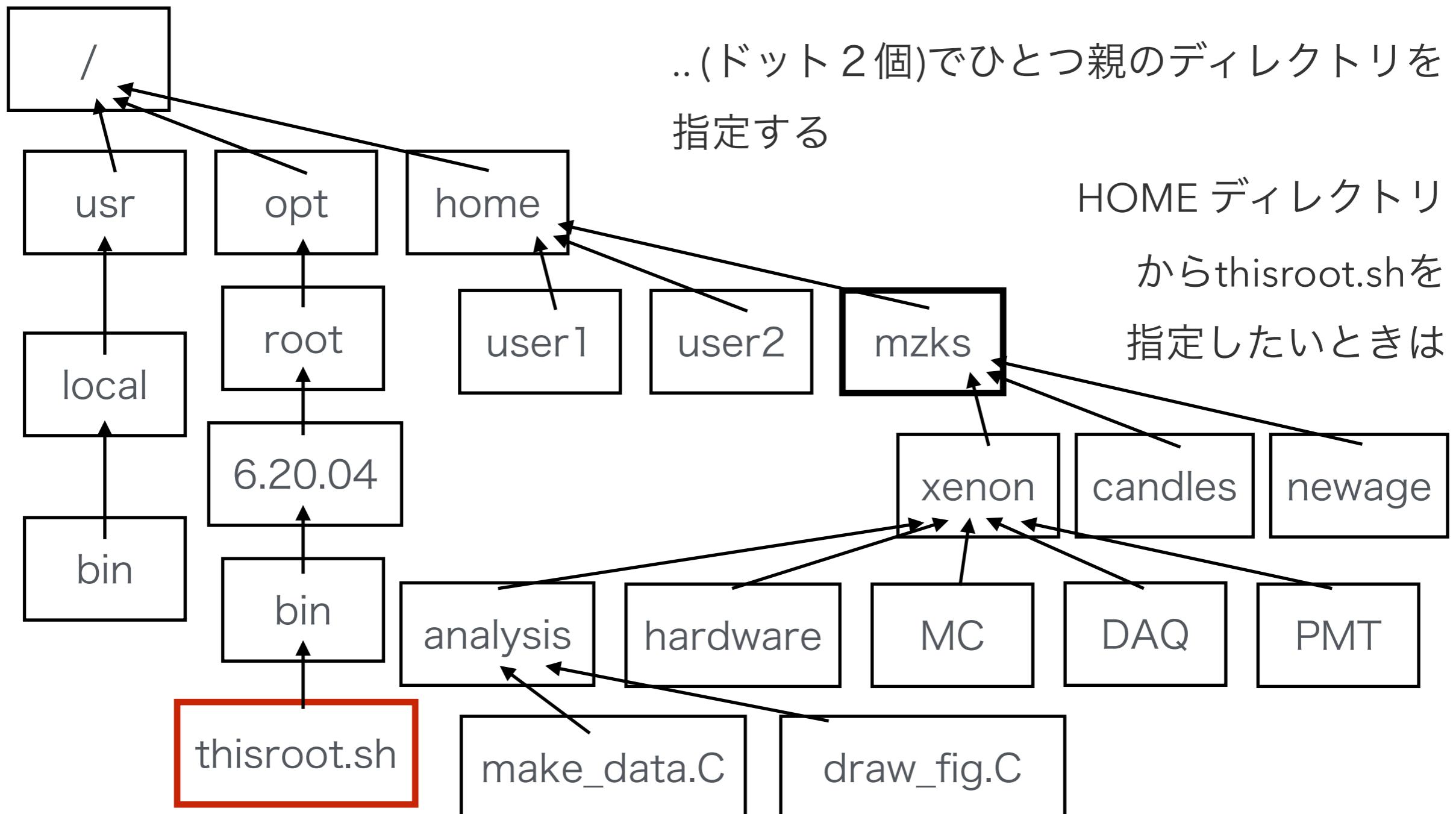
# HOME ディレクトリ



HOME ディレクトリ  
から make\_data.C を  
指定したいときは

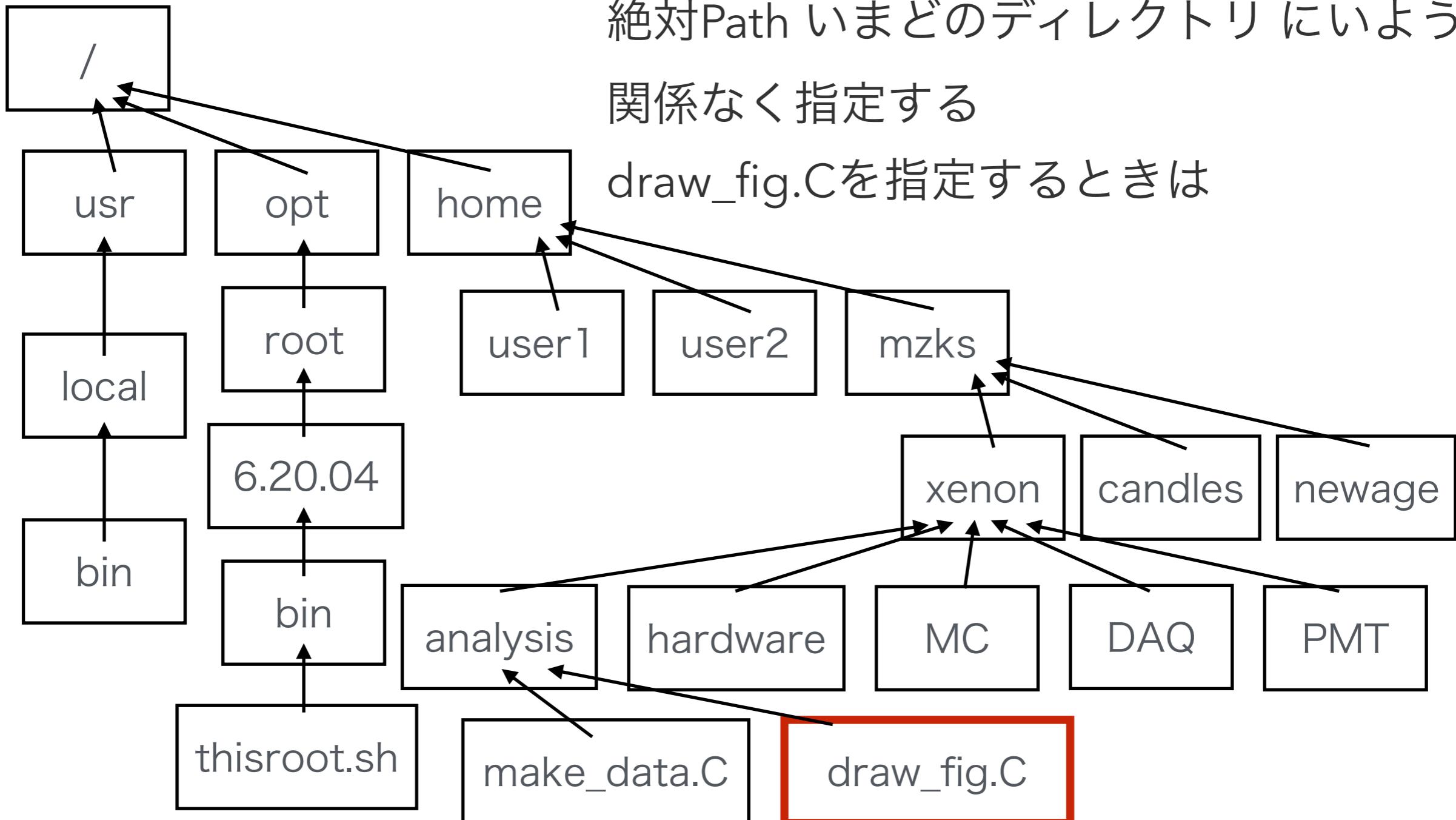
- xenon/analysis/make\_data.C と指定
- ディレクトリ の区切り文字は'/'

# 親のディレクトリを指定



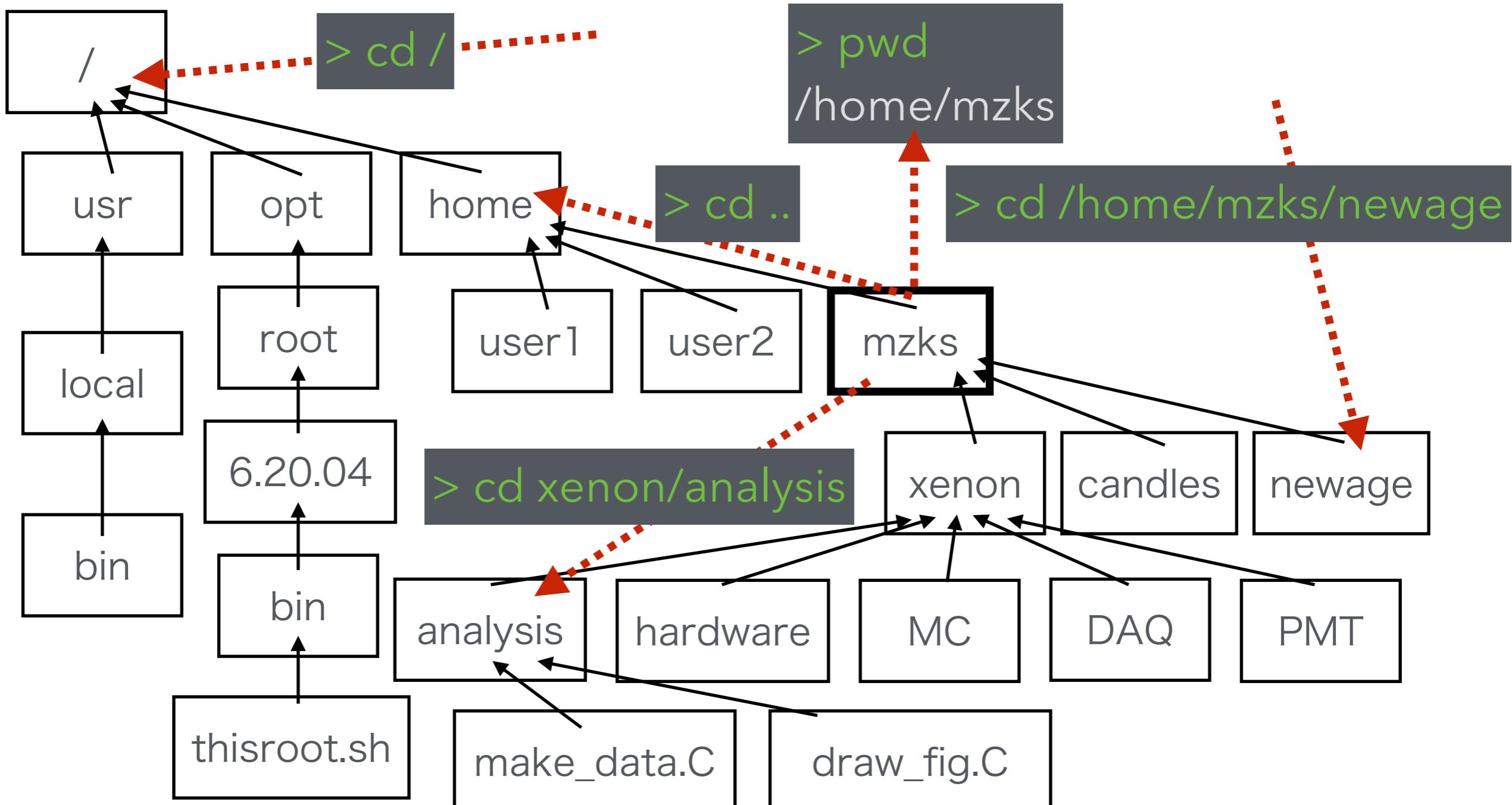
- `../../opt/root/6.22.08/bin>thisroot.sh` と指定する

# 絶対Path



- /home/mzks/xenon/analysis/draw\_fig.C

# cd コマンド – change directory



- 現在のディレクトリを変更するコマンド
- 絶対パスで指定すると現在のディレクトリに関係なく移動する

# ファイル操作とパス その1

- Path – Linux (Unix)計算機でファイルの場所を示す
- ディレクトリ(フォルダ)の入れ子
- ディレクトリの区切りは'/'
- 絶対パス
  - ファイルシステムの最初(/)からファイルを指定
  - 例: /home/mzks/root\_lecture/first\_macro.C
- 相対パス
  - 今いるdirectory (current directory)からみてファイルを指定
  - 例: ./root\_lecture/first\_macro.C
- コマンドlsで現在のディレクトリにあるファイルが見える

```
>cd  
>pwd  
/Users/mzks/ ←現在いるディレクトリ (Home directory)  
>ls  
Desktop Documents Downloads xenon
```

# ファイル操作とパス その2

- ディレクトリやファイルを作ったり消したり
  - **mkdir directory\_name** :directory\_nameのなまえの directoryをつくる
  - **mv file\_name1 file\_name2** :file\_name1をfile\_name2に移動する(名前を変える)
  - **rm -r file\_name**  
:file\_nameを削除する⚠
- 作ったり消したりして操作になれてください

```
>cd  
>pwd  
/Users/mzks/  
>mkdir root_lecture  
>cd root_lecture  
>pwd  
/Users/mzks/root_lecture  
>mkdir day1  
>cd day1  
>pwd  
/Users/mzks/root_lecture/day0  
>cd ..  
>pwd  
/Users/mzks/root_lecture  
>ls  
day1  
>cd ~/root_lecture
```

# Tabキー習慣

- Tabキーを有効活用しよう
- コマンドやファイルパスを途中まで打っている状態でTab
- 残りが自動で補完される
- 候補が複数あるときはあつているところまで
- 候補も表示する
- 逆に表示されなければこれまで入力した部分が間違っているということ
- 押すことでこれまでの入力を変えてしまうことはないので安心して使う

```
>cd  
>cd ro ← ここでroまでうつてTab  
  
>cd root_lecture/ ← 残りが勝手に入る  
  
>cd root_lecture/ ← ここでTab  
macros/ shell_bench/  
  
>cd root_lecture/m ← mだけうつてTab  
  
>cd root_lecture/macros/ ← 残りが勝手に入る  
← ここでEnterを押してコマンド実行  
  
>ls  
first_macro.C function_macro.C  
  
>cat fi ← fiまで打つと候補は一つしかない  
← Tabを押すとこの候補に補完される  
  
>cat first_macro.C  
(出力省略)
```

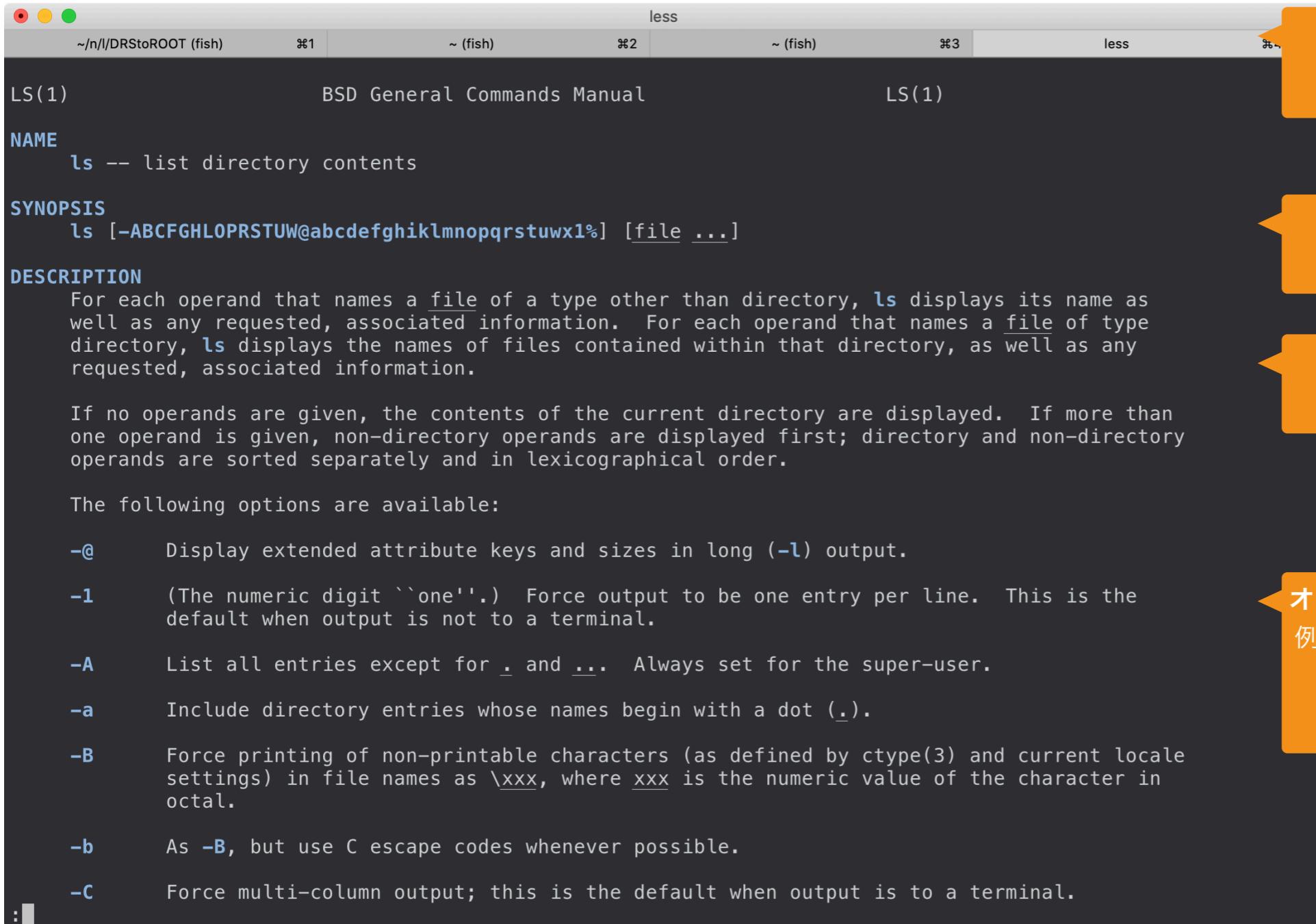
# ls -l (list コマンド)

```
>ls  
first_macro.C      function_macro.C  
  
>ls -l  
total 16  
-rw-r--r--  1 mzks  staff   104 Apr  9 18:49 first_macro.C  
-rw-r--r--  1 mzks  staff    42 Apr  9 22:37 function_macro.C
```

- lsコマンドは現在のディレクトリに含まれているファイル、ディレクトリを表示するコマンド
- ls -l ('l', 's', 'スペース', '-' , 'l')の順で打つとより詳細な情報を表示
- このように、コマンドの後に '-' をつけて、コマンドの挙動を制御するものをオプションと呼ぶ
- lsコマンド以外にも、ほぼ全てのコマンドにオプションがある

# man コマンド

- manコマンドはコマンドのmanualを表示するコマンド
- 使い方を忘れたコマンドはここで確認
- 初めはぐぐることの方が多いかもしれないが、癖をつけておくと後々役に立つ



terminalで `>man ls` とした結果  
jkで上下移動, qで終了

NAME  
`ls` -- list directory contents

SYNOPSIS  
`ls [-ABCFGHLOPRSTUW@abcdefghijklmnopqrstuvwxyz1%] [file ...]`

DESCRIPTION  
For each operand that names a `file` of a type other than directory, `ls` displays its name as well as any requested, associated information. For each operand that names a `file` of type directory, `ls` displays the names of files contained within that directory, as well as any requested, associated information.

If no operands are given, the contents of the current directory are displayed. If more than one operand is given, non-directory operands are displayed first; directory and non-directory operands are sorted separately and in lexicographical order.

The following options are available:

- `-@` Display extended attribute keys and sizes in long (`-l`) output.
- `-1` (The numeric digit ``one''.) Force output to be one entry per line. This is the default when output is not to a terminal.
- `-A` List all entries except for `.` and `...`. Always set for the super-user.
- `-a` Include directory entries whose names begin with a dot (`.`).
- `-B` Force printing of non-printable characters (as defined by `ctype(3)` and current locale settings) in file names as `\xxx`, where `xxx` is the numeric value of the character in octal.
- `-b` As `-B`, but use C escape codes whenever possible.
- `-c` Force multi-column output; this is the default when output is to a terminal.

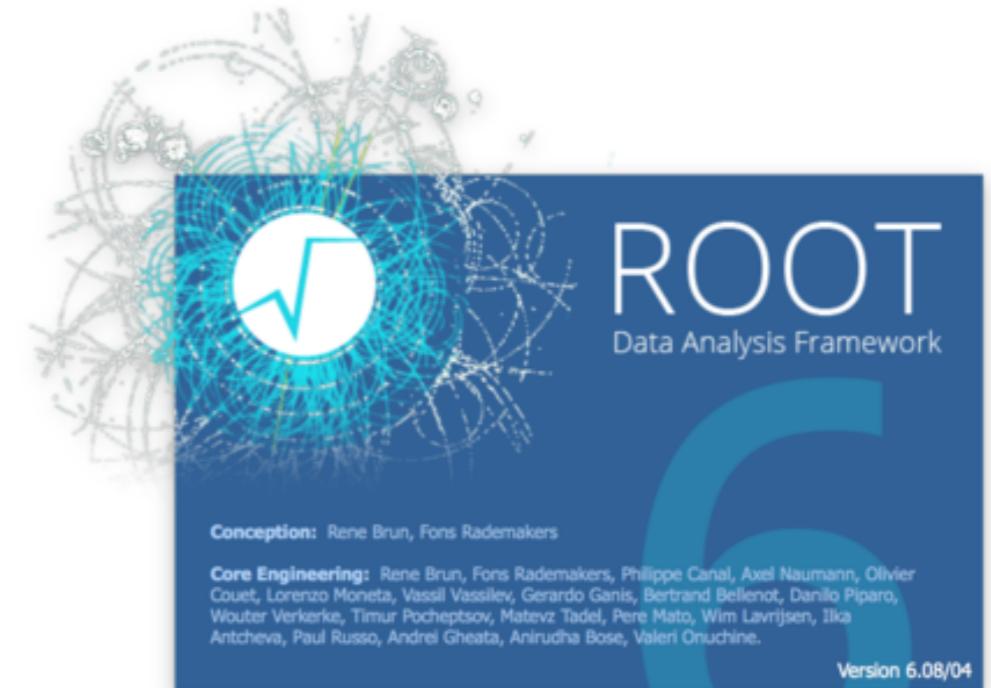
ここに使い方が書いてある  
[-AB..]とかは使えるオプション

コマンドの説明

オプションの説明  
例えば -aはドットで始まるファイルも表示  
するようにするオプション

# コマンドラインからROOTを起動

- コマンドラインでroot
- 右のスプラッシュが表示され,rootのプロンプトが起動
- Shellではなく,rootに命令できる
- root[なんか数字]の後に  
C++の文を入力すると1行ずつ実行する.



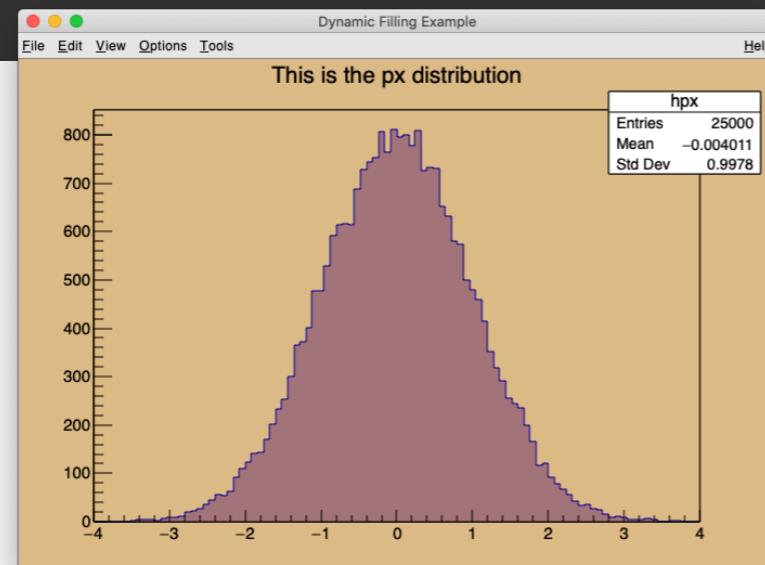
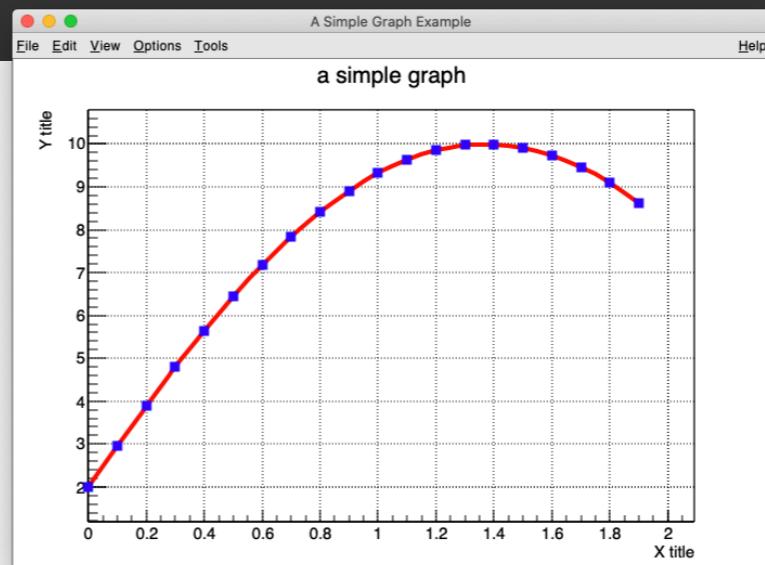
```
% root
-----
| Welcome to ROOT 6.16/00          https://root.cern |
| (c) 1995-2018, The ROOT Team   |
| Built for macosx64 on Jan 23 2019, 09:06:13 |
| From tags/v6-16-00@v6-16-00    |
| Try '.help', '.demo', '.license', '.credits', '.quit'/.q' |
|-----|
root [0] cout << "Hello, ROOT" << endl;
Hello, ROOT
root [1] 1+1
(int) 2
root [2]
```

空気を読んで文末のセミコロン ; が  
なくても動く

# 簡単すぎて退屈な人向け

- rootの機能デモをみて、できることを眺めてみてください
  - rootを起動して.demoとうつ

```
% root
-----
| Welcome to ROOT 6.16/00          https://root.cern
| (c) 1995-2018, The ROOT Team
| Built for macosx64 on Jan 23 2019, 09:06:13
| From tags/v6-16-00@v6-16-00
| Try '.help', '.demo', '.license', '.credits', '.quit'/.q'
-----
root [0] .demo



```

# ROOT Tutorialの追加インストール(必要な人)

- ・`brew install root` などでROOTをインストールして、  
Tutorialがない場合は以下のコマンドで取得する

```
% cd root_lecture  
% wget https://root.cern/download/root_v6.26.00.source.tar.gz  
...  
% tar xzvf root_v6.26.00.source.tar.gz  
...  
% cd root-6.26.00/tutorials/  
% root
```

# マクロ

- ・何度もROOTを立ち上げてコマンドを打ち込むのは面倒だ
- ・ファイルにコマンドを記述して,ROOTに実行させることができる
- ・実行コマンドを一行ずつファイルに記述して,{}で囲む
- ・コマンドラインで実行
  - >root first\_macro.C
- ・マクロを書いて実行→一部修正,追記→実行 を繰り返して解析を進めていく
- ・C++はmain関数から処理がスタートするが,{}で括るROOTマクロは単に上から実行される

```
first_macro.C
1 {
2
3     cout << "Hello, World!" << endl;
4
5     int val = 1+1;
6     cout << "val = " << val << endl;
7
8 }
```

# タッチタイピング

- タイピング速度は非常に重要
  - 人生の時間を節約できる
- タッチタイピング(=キーボードを見ないで打つ)は必須
  - 画面を見続けることができる ← 肩こり防止
- 初めは大変だが、1週間やると劇的に改善する
- 練習サイト
  - 寿司打 <http://typingx0.net/sushida/>
  - e-typing <https://www.e-typing.ne.jp>
- 練習ツール
  - typro <https://pypi.org/project/typro>
  - コンソールで pip install typro でインストールできる
  - typro -f root とか。

# Day1 宿題

- 来週のために、好きなエディタをインストール

してきてください。

- Vim, Emacs, VSCode, Atom, 等...

なんでもいいです

- VimとEmacsは初心者には少し難しい

かもしれません

- Shellのコマンドを調べてあそんでみてください

- ROOTのプロンプトでC++の文を実行して

遊んでみてください

- C++の復習になると思います

- タッチタイピング練習をしてください。



```
root [0] for(int i=1;i<100;++i){  
root (cont'ed, cancel with .@) [13] if(i%3==0 && i%5==0) cout << "Fizzbuzz"  
<< endl;  
root (cont'ed, cancel with .@) [14] else if(i%3==0) cout << "Fizz" << endl;  
root (cont'ed, cancel with .@) [15] else if(i%5==0) cout << "Buzz" << endl;  
root (cont'ed, cancel with .@) [16] else cout << i << endl;  
root (cont'ed, cancel with .@) [17] }  
1  
2  
Fizz  
4  
Buzz  
Fizz  
7  
8  
Fizz  
以下略
```

# おまけ: typro

- 最近暇つぶしにつくったタイピングゲーム
- python3が入っていれば、コンソールで``pip install typro``でインストールできる。
- ``pip3 install typro``かも。
- WSLは初回にPATHを通す必要があるかも
  - `echo 'export PATH="$PATH:~/local/bin"' >> ~/.bashrc`
- 遊んで(練習してみて) ください。