高エネルギー宇宙物理学のための ROOT 入門

– 第 1 回 –

奥村 曉

名古屋大学 宇宙地球環境研究所

2017 年 4 月 25 日

名古屋大学 NAGOYA UNIVERSITY

ISEE Institute for Space-Earth Environmental Research

oxon@mac.com

はじめに

簡単な自己紹介

- 名古屋大学の宇宙地球環境研究所（ISEE）で講師をしています
- 最近は主に Cherenkov Telescope Array（CTA）のハード開発をしています
- ソフトウェアの専門家ではない
- ROOT の専門家でもないが
  - ROBAST（https://robast.github.io/）という ROOT を使った光線追跡シミュレーションのライブラリを宇宙線望遠鏡向けに開発したり
  - CTA の DAQ ソフトウェアの開発をしたり
- 主な使用言語：C++ と Python

宇宙地球環境研究所（ISEE）の紹介

宇宙地球環境研究所
Institute for Space–Earth Environmental Research

- Institute for Space–Earth Environmental Research（**ISEE**）
- 名古屋大学の附置研究所のひとつ（旧太陽地球環境研究所が他 2 センターと統合）
- 宇宙線研とともに、宇宙線分野の研究も含む**共同利用・共同研究拠点**
- 基礎研究部門のうち、**宇宙線研究部**が複数の宇宙線実験プロジェクトを行なっている
  - CTA、Fermi/LAT、LHCf/RHICf、SK/HK、XMASS/XENONnT、太陽中性子望遠鏡、屋久杉（^{14}C）、MOA 望遠鏡などを幅広く（教員 9 名）
  - http://www.isee.nagoya-u.ac.jp/CR/
- そのうち YMAP 研究会を ISEE でも開きたい（来年度？）

講習の進め方

- 参考書『高エネルギー宇宙物理学のための ROOT 入門』
  - 2009 年からダラダラと執筆、まだまだ完成には程遠い（最新版 v1.5）
  - GitHub から LaTeX ファイル一式、もしくは PDF を落とせる
    - https://github.com/akira-okumura/RHEA
    - https://github.com/akira-okumura/RHEA/releases
    - Wiki も参照のこと
      - https://github.com/akira-okumura/RHEA/wiki/ROOT-講習会-2018
- 毎週木曜日の 17:00〜18:30 で合計 5 回（例年は 6 回）
  - 4/26、5/10、5/24、5/31、6/7
  - 出張と GW で飛び飛びです
- 面白くなければ、内職、退席ご自由にどうぞ（うちの学生以外）
- 飲食自由

注意とお願い

- 多地点中継は昨年度色々とトラブルが発生しますが、ご容赦ください
- 「言われた通りに動かない」「説明の意味が分からない」という場合は、必ず講師もしくは各大学の先輩に**即座に質問**してください。
- 例年、講義の後から質問してきたり、分からないままついていけなくなる事態が発生しています
- 講師側からは中継先の状況が見えず、壁に向かって話しかけているような状態なので、講師側からの質問には必ず応答してください。（例：「聞こえてますか？」「はい」）
- 名大の宇宙線研究室の学生向けに始めた身内の講習会を、ボランティアで YMAP 向けに中継するものです。参加は強制ではないので、あまり参加意欲のない方は抜けてください。

質問について

- 日本の教育では、講義中に質問をするという自然な雰囲気がなぜか醸成されておらず、多くの大学生は質問慣れしていない
- 「こんな質問したら理解の早い他の人には迷惑かも」「講義を中断させてしまう」といった心配は捨てましょう
- 参加者は 100 人います。あなたの質問は、他の人の疑問解消にもきっと役立ちます。
- 講師の説明に間違いがあったら遠慮なく指摘しましょう
- 質問力、質問慣れは、大学院で必要になります

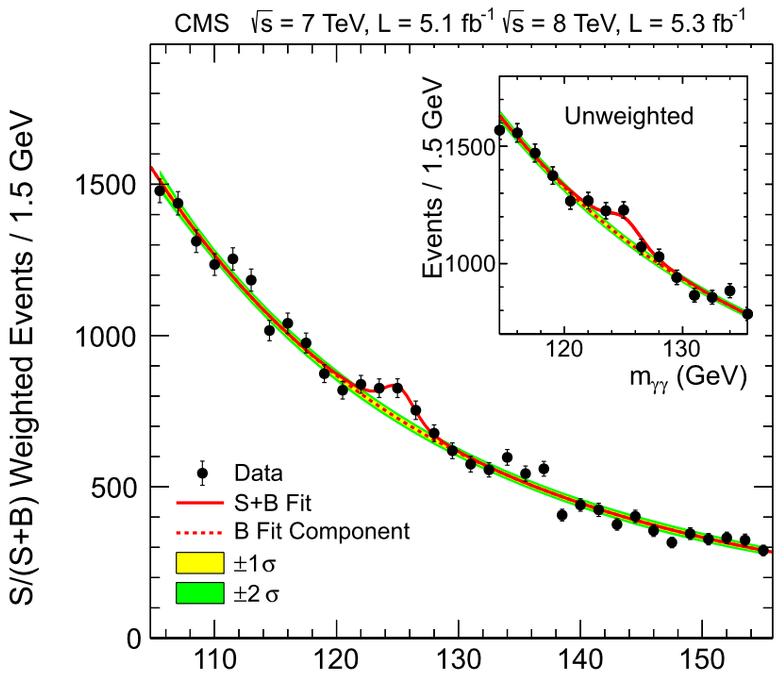
講習の目的

- 日本ではソフトウェアは軽視されており、大学院で使える実践的なプログラミングの講義を受ける機会は少ない
- そもそも、ちゃんと教えられる人が業界内に少ない
- 中途半端な知識を先輩から受け継いでも科学は進歩しない
- 毎年毎年、新入生に同じことを伝えるのは時間の無駄
- 大学ごとに同じことを平行してやるのも時間の無駄
- 宇宙線業界の標準解析ツールである ROOT と、標準言語である C/C++/Python の敷居を下げる
- ソフトウェア全般についての知識を増やす
- 計算機を使った研究の入り口として、奥村がまず知っておいて欲しいと思うことを中心に取り上げる

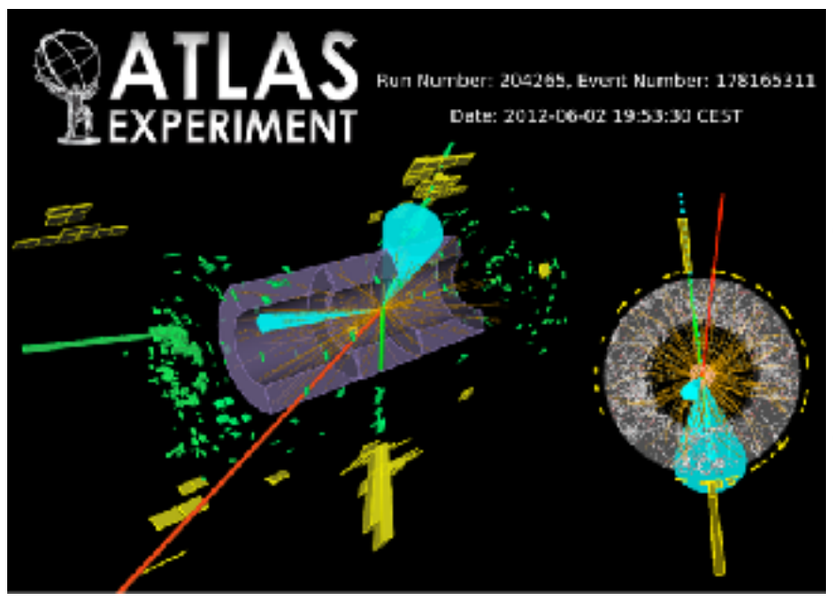
ROOT ってなに？

- 実験系素粒子物理学では標準的なデータ解析ソフトウェア・ライブラリ群
- 他の解析ソフトとなにが違うか？
  - 文法が C++ なので学習が容易（R/gnuplot のように独自文法を覚えなくて良い）
  - そのまま C++ ソフトウェアに組み込める
  - ヒストグラムなど物理実験に必須のデータ解析ツールを備える
  - Tree（ツリー）というデータ構造があり、イベントの概念を持つ物理実験で非常に強力（そのうち説明）
  - もともとが素粒子実験向けに作られているので、我々の欲しい機能が詰まっている（そして追加される）
  - ただし天文系の解析には弱い → Python との併用/移行が主流
- ROOT を主流派としない宇宙線プロジェクトも増えてきている
  - IceCube、CTA、XENONnT など（Python と matplotlib）
  - 論文の図を見比べると判る

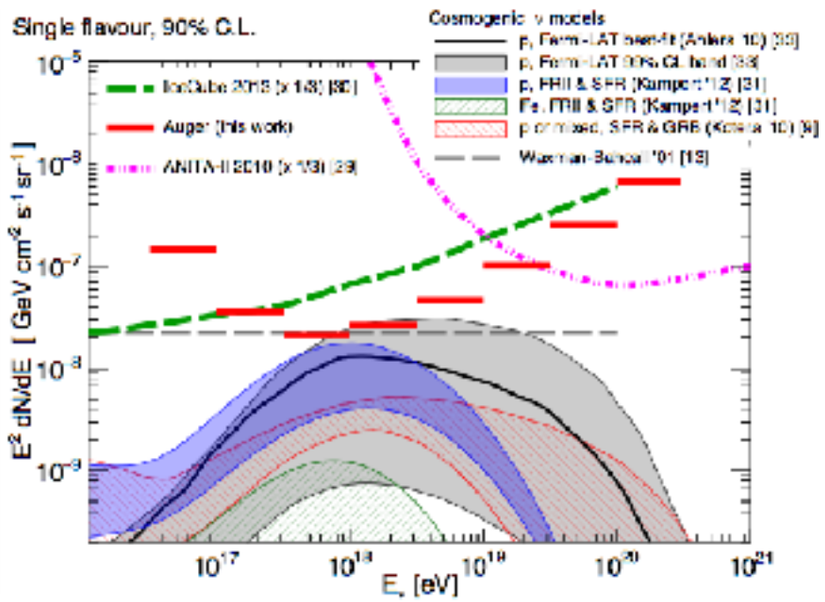
ROOT の使用例（有名な例 + TGraph + TH1 + 手前味噌）



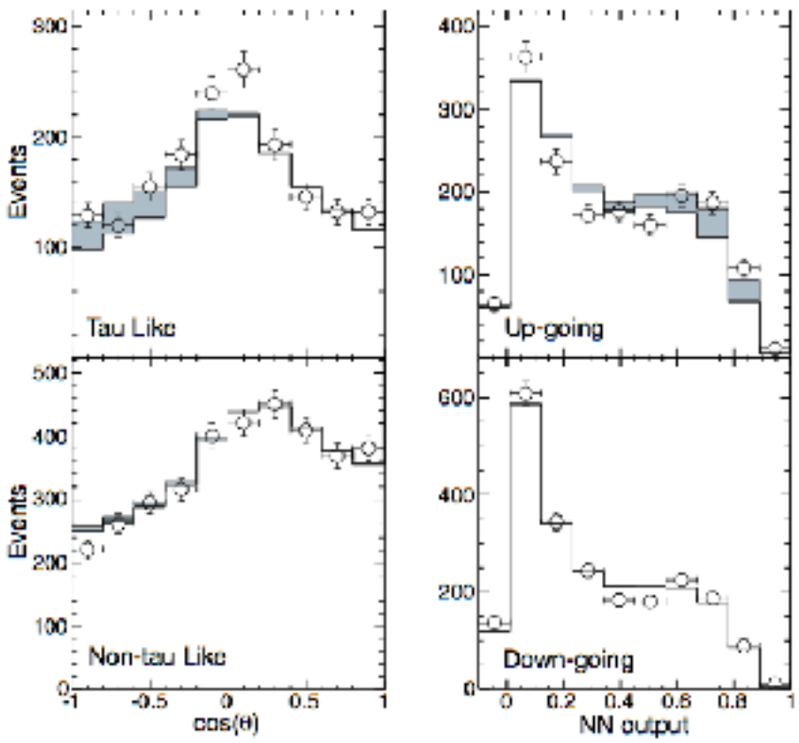
Chatrchyan+ 2012（CMS）



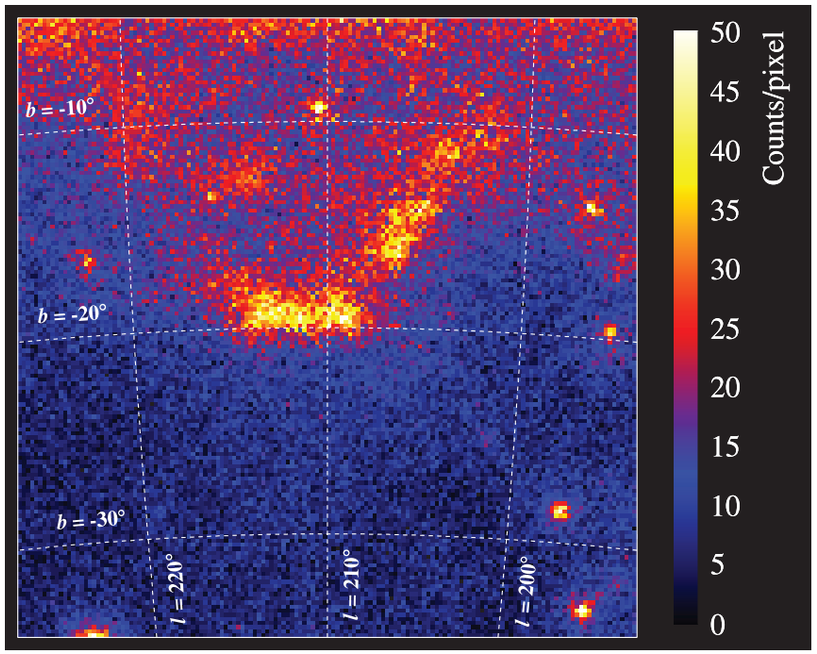
Credit: ATLAS



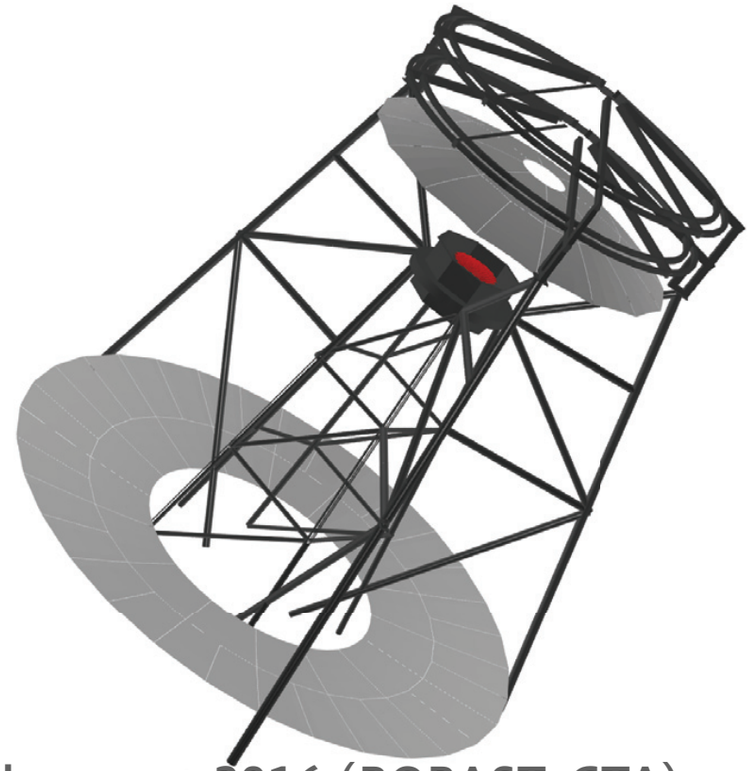
Aab+ 2015（IceCube）



Abe+ 2013（SK）



Ackermann+ 2012（Fermi/LAT）



Okumura+ 2016（ROBAST, CTA）

言語の違い

- Fortran
  - 理論屋さんと一緒に計算をするのでもない限り、若手の会参加者は忘れてしまってよい
  - 指導教員が Fortran 使うように言ってきたら無視するように
- C 言語
  - 様々な OS や「枯れた」ライブラリは C で書かれており、豊富なシステムコールを持つ
  - 宇宙線屋が使うのは、主にハードウェア制御のときなど
  - 言語自体は簡単だけど、実際に使われるのは複雑なプログラムの中なので、初心者には取り組みにくい
- C++
  - 「オブジェクト指向」という思想を持った C の拡張
  - ROOT や Geant4 は C++ で書かれている（ただし ROOT のソースは現代的ではないので、あまり参考にしないほうがよいし、Geant4 はめちゃくちゃ汚い）
  - 宇宙線業界ではデータ解析、ハードウェア制御、シミュレーションで広く使われる
- Python
  - インタプリタ言語（コンパイルしなくて良い）なのですぐに簡単に使える
  - 色々なライブラリが標準で備わっており、C++ より多用途
  - M1 が学習するには多分 C++ よりも簡単
  - PyROOT を使って、Python 内でも ROOT を使える（自分は普段はこれ）

第 1 回の前提

- コンパイラ等の環境が整った、macOS もしくは Linux（CentOS 7.4 を仮定）の走る計算機が手元にあること
  - GCC（Linux の場合）もしくは Clang（macOS の場合）が入っている
  - CMake 3.x が入っている（Yum もしくは Homebrew）
  - Python 3 が入っている（推奨、Python 2 でも可）
  - IPython が入っている（強い推奨、普通の python コマンドでも可）
- GitHub のアカウントを作る
  - 参考書の最新版が GitHub にあるため
  - ソフトウェア開発では、今後 GitHub は避けて通れない
  - 「レポジトリ」「バージョン管理」という概念に触れる
  - 改訂の要望や講習会後の質問は GitHub 経由でやってもらう
  - ただし、これまでは血気溢れる学生が少なく、GitHub は活発に使われなかった
- 日本語 LaTeX 環境があるのが望ましい
  - 参考書の最新版を自分で PDF にコンパイルするため
  - どうせ夏の学校の収録や修論で必要になります

第 1 回の前提（続き）

- ROOT 6.12/06 のインストール
  - 参考書の 2.1 節、2.2 節を参照
  - 読めば分かるし、講習中にやるのは時間の無駄
  - ROOT 5 でも大差ないが、ROOT 6 環境を前提とする
    - ROOT 6 のほうが C++ の文法に厳格（間違った文法を覚えない）
    - エラーの表示が分かりやすい
    - 「標準偏差」と「RMS」の用語混同が直った（20 年近く間違っていた）
    - 計算機上に ROOT 5 と 6 を両方入れても大きな問題は起きない（ただし、自分で環境変数をちゃんと管理すること）
- ソースからビルドすること（Clang や GCC の動作確認のため）

GitHub

GitHub ってなに？

- オープンソースソフトウェア開発で、ほぼ標準の地位を獲得したサービス
- Git というソフトウェアでバージョン管理を行う
  - OS X でも Linux でも、多分最初から入ってる
  - Linux に入ってなければ yum で入れる
- 例えば Astropy や ROBAST も GitHub 上でコード開発をしてい
  - http://www.astropy.org/
  - https://github.com/ROBAST
- README.md を読む
- 今はあまり細かいことは知らなくて良い

Git コマンドを使ってみる、PDF を生成してみる

```
$ git clone https://github.com/akira-okumura/RHEA.git
```
❶ レポジトリをクローンする

※ ここで git コマンドがあるかどうか分かる

```
$ cd RHEA
$ ls
Makefile  RHEA.bib  RHEA.tex  misc      tex
README.md RHEA.bst  fig       src
$ make
```
❷ 中身を確認

※ make および日本語 LaTeX 環境必須

❸ make を実行

```
$ ls
Makefile  RHEA.bbl  RHEA.bst  RHEA.out  RHEA.toc  src
README.md RHEA.bib  RHEA.dvi  RHEA.pdf  fig       tex
RHEA.aux  RHEA.blg  RHEA.log  RHEA.tex  misc
```
❹ RHEA.pdf の生成を確認

```
$ open RHEA.pdf
```
※ macOS の場合

❺ RHEA.pdf を開く

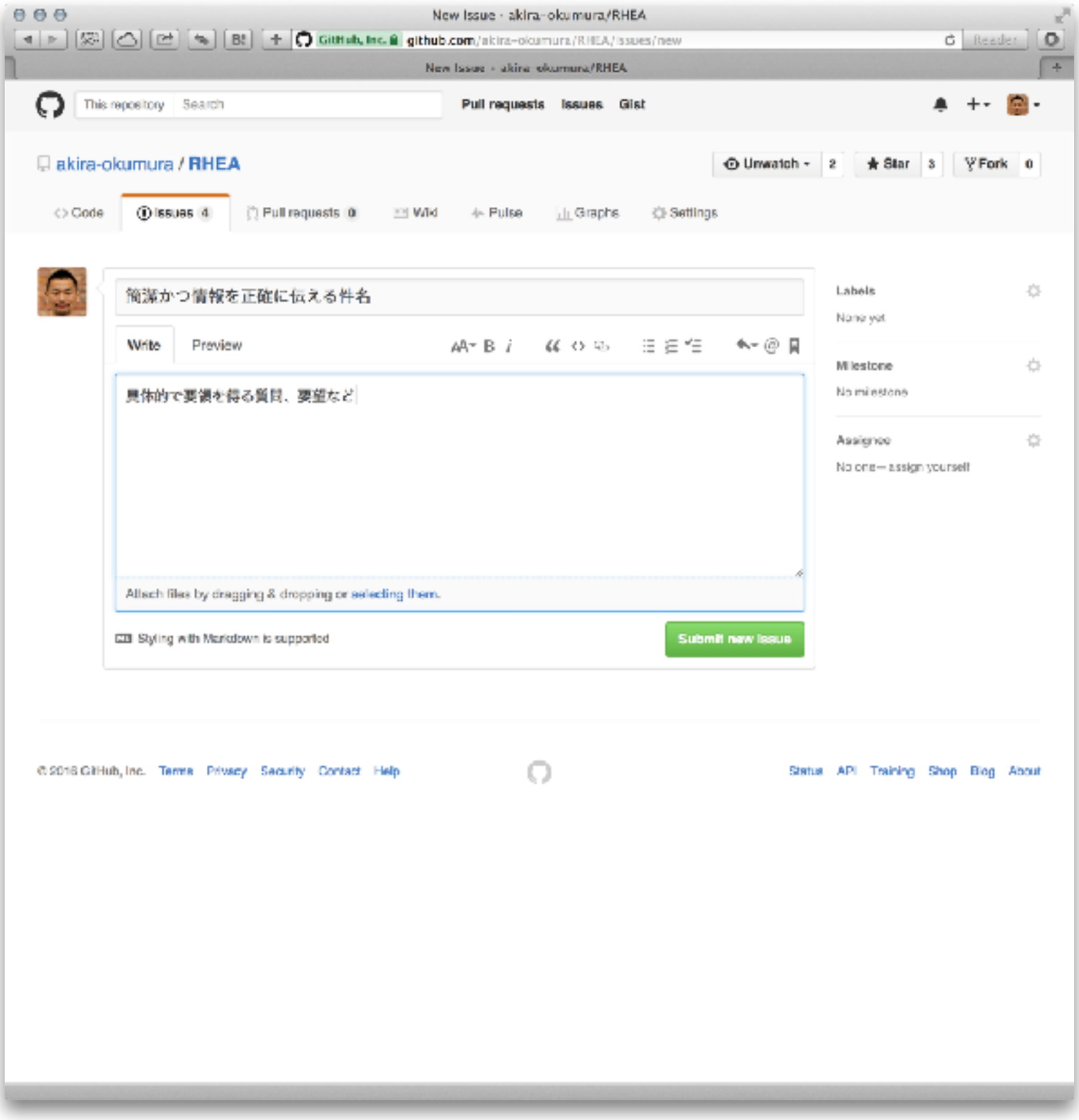
```
$ xdg-open RHEA.pdf
```
※ Linux の場合

RHEA レポジトリの最新版へ更新する

```
$ cd RHEA
$ git pull
$ make
$ open RHEA.pdf
$ xdg-open RHEA.pdf
```

❶ レポジトリの最新版を取得
❷ 再度 PDF を生成する

GitHub 上での質問、改訂の要望



- GitHub から新規 issue を作り質問や改訂の要望を出す
  - https://github.com/akira-okumura/RHEA/issues/new
- 件名は簡潔かつ正確に
- 内容は具体的かつ要領を得る書き方を
  - 問題を再現する最小構成のスクリプトなど
  - 計算機環境
  - ROOT のバージョン
  - 参照した参考書の版

ROOT のビルド

(念のため)

ROOT のビルド（configure を使った古いやり方）

```
$ cd ~
$ curl -O https://root.cern.ch/download/root_v6.08.06.source.tar.gz
$ cd /usr/local
$ sudo tar zxvf ~/root_v6.08.06.source.tar.gz

$ cd root-6.08.06
$ sudo ./configure
（省略）
Enabled support for asimage, astiff, bonjour, builtin_afterimage, builtin_ftgl,
builtin_freetype, builtin_glew, builtin_llvm, libcxx, cocoa, explicitlink, fink,
fftw3, fitsio, gviz, genvector, krb5, ldap, mathmore, memstat, opengl, python,
rpath, search_usrlocal, shared, sqlite, ssl, tmva, vdt, xml.

To build ROOT type:

   make
$
```

❶ ソースコードを落としてくる
❷ /usr/local に展開

※ sudo が使えなければ、システム管理者に
相談するか、他の場所に展開する

❸ configure を実行

※ ここまで出れば configure は成功

※ ROOT 6.09.02 から configure + make はちゃんと動かないようです

ROOT のビルド（configure を使った古いやり方）

```
$ sudo make -j 8                                          ❹ make を実行
（省略）
root [0]
Processing hsimple.C...
hsimple   : Real Time =   0.07 seconds Cpu Time =   0.05 seconds
(TFile *) 0x7fb0fae560c0

   ====================================================================
   ===                  ROOT BUILD SUCCESSFUL.                      ===
   === Run 'source bin/thisroot.[c]sh' before starting ROOT ===
   ====================================================================
$
```

※ ここまで出ればビルド成功

ROOT のビルド（管理者権限でやる場合）

```
$ cd ~
$ curl -O https://root.cern.ch/download/root_v6.12.06.source.tar.gz
$ cd /usr/local
$ sudo tar zxvf ~/root_v6.12.06.source.tar.gz

$ cd /usr/local/root-6.12.06
$ mkdir cmake_build
$ cd cmake_build

$ sudo cmake ..
$ sudo cmake3 ..
（省略）
-- Configuring done
-- Generating done
-- Build files have been written to: /usr/local/root-6.12.06/cmake_build
```

❶ ソースコードを落としてくる
❷ /usr/local に展開

❸ build の場所を用意

❹ CMake を実行　※ Homebrew　※ Yum

※ ここまで出れば CMake は成功

ROOT のビルド（管理者権限でやる場合）

```
$ sudo make -j 8                                          ❺ make を実行
（省略）
[100%] Generating tutorials/hsimple.root
Processing hsimple.C...
hsimple   : Real Time =   0.06 seconds Cpu Time =   0.05 seconds
(TFile *) 0x7f9f7cda9760
[100%] Built target hsimple

                                                  ※ ここまで出ればビルド成功
```

ROOT のビルド（一般ユーザ権限 = sudo を使わない場合）

```
$ cd ~
$ curl -O https://root.cern.ch/download/root_v6.12.06.source.tar.gz
$ tar zxvf ~/root_v6.12.06.source.tar.gz

$ cd root-6.12.06
$ mkdir cmake_build
$ cd cmake_build

$ cmake ..
$ cmake3 ..
（省略）
-- Configuring done
-- Generating done
-- Build files have been written to: /usr/local/root-6.12.06/cmake_build
```

❶ ソースコードを落としてくる
❷ ~/ に展開

❸ build の場所を用意

❹ CMake を実行

※ Homebrew
※ Yum

※ ここまで出れば CMake は成功

ROOT のビルド（一般ユーザ権限 = sudo を使わない場合）

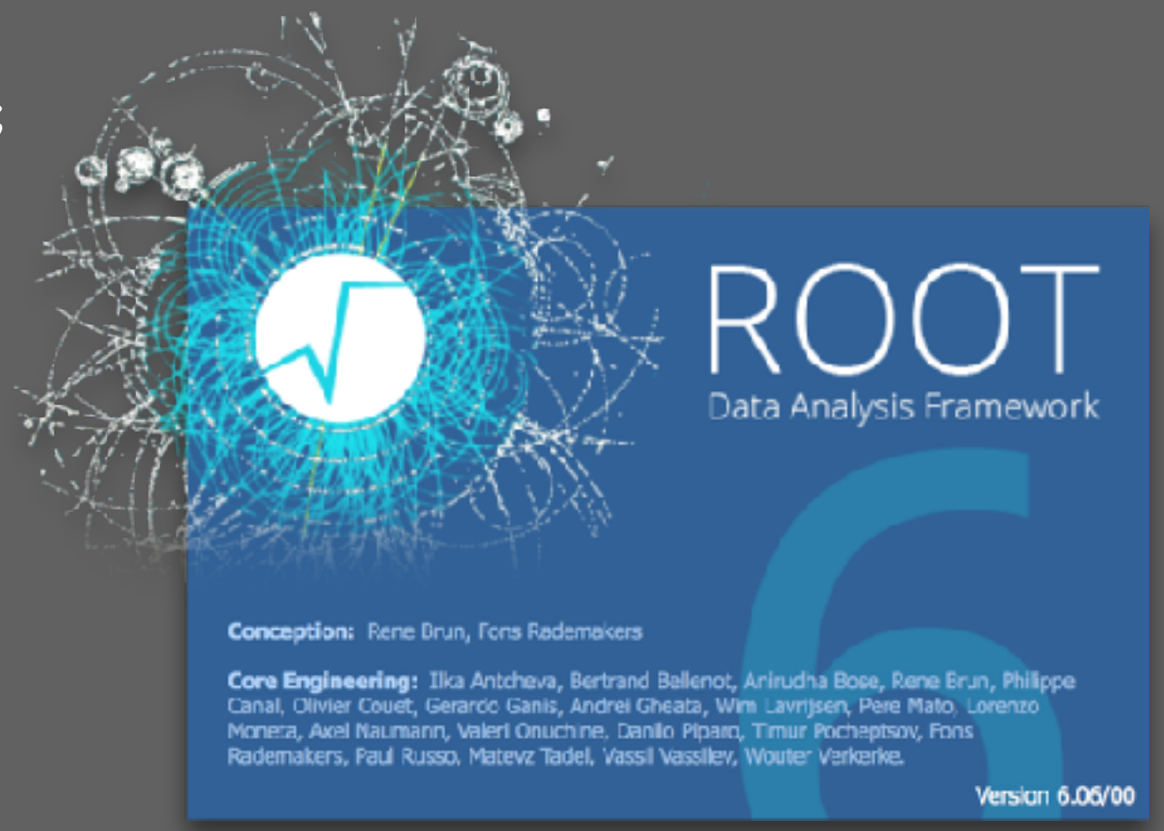
```
$ make -j 8                                                    ❺ make を実行
（省略）
[100%] Generating tutorials/hsimple.root
Processing hsimple.C...
hsimple   : Real Time =   0.06 seconds Cpu Time =   0.05 seconds
(TFile *) 0x7f9f7cda9760
[100%] Built target hsimple

                                                  ※ ここまで出ればビルド成功
```

環境変数の設定（bash や zsh の場合）

```
$ cat ~/.bashrc                                  ※ zsh の場合は .zshrc に置き換える
cd /usr/local/root-6.12.06
source bin/thisroot.sh                           ※ .bashrc の中身（ROOT 用の最小設定）
cd - > /dev/null

$ source ~/.bashrc                               ※ 初回だけ必要

$ echo $ROOTSYS                                  ※ 環境変数がいくつか追加される
/usr/local/root-6.12.06/cmake_buiid
$ echo $PATH
/usr/local/root-6.12.06/cmake_buiid/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/
local/bin:/opt/X11/bin
                                                 ※ root コマンドは $ROOTSYS/bin 以下に
```

ROOT の起動と終了

```
$ root
   ------------------------------------------------------
  | Welcome to ROOT 6.12/06                  http://root.cern.ch |
  |                                (c) 1995-2017, The ROOT Team |
  | Built for macosx64                                          |
  | From tag v6-12-06, 9 February 2018                          |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q' |
   ------------------------------------------------------

root [0] std::cout << "Hello World!\n";
Hello World!
root [1] .q
$
```




```
$ alias root="root -l"
$ root
root [0]
```

※ 余計なものが消える

Python 3 を使う場合

```
$ sudo cmake .. -DPYTHON_EXECUTABLE=/usr/local/bin/python3
```

※ **Homebrew**

```
$ sudo cmake3 .. -DPYTHON_EXECUTABLE=/usr/bin/python3.6
```

※ **Yum**

```
（省略）
-- Configuring done
-- Generating done
-- Build files have been written to: /usr/local/root-6.12.06/cmake_build
```

configure と make ってなに？

- configure（付録 C 参照）
  - Linux や macOS でソースコードからソフトウェアをビルドするときに頻繁に目にするコマンド（Autotools というソフトが configure コマンドを生成）
  - 計算機環境に応じて、ビルドに必要な他のライブラリの有無を調べたり、OS 特有の設定をしてくれたりする
  - 色々とビルドのオプションを指定することも可能
    例えば ./configure --prefix=/opt/local など
  - Makefile を生成する
- make
  - Makefile に書かれた内容にしたがって、多数のソースコードを順次コンパイル、リンクする
  - ROOT であれば、ROOT の各種ライブラリと実行ファイルを生成
  - $ROOTSYS/bin/root や $ROOTSYS/lib/* が作られる
  - 単純な例 https://github.com/akira-okumura/RHEA/blob/master/Makefile
  - 他のソフトウェアでは make install というコマンドも必要だが、ROOT では不要
- 最近はパッケージ管理システム（Homebrew や Yum）が整っているので、使わなくても生きていける場合が多い（付録 A 参照）

CMake

- ROOT や Geant4 では configure + make から **CMake** に移行した
- CTA でも CMake の使用が推奨されている
- 利点は Windows でも動き、文法がわかりやすいこと
- 欠点は転がっている情報がまだ少ない
- 最近の ROOT だと、configure + make は動作しなくてもサポートされないので、CMake 推奨
- CMake 3.4.3 以上が必要なので、Scientific Linux 7 などだと標準の CMake が 2.8 なので、自分で 3.4 以上を入れないといけない

CMake でのビルド

```
$ cd ~
$ curl -O https://root.cern.ch/download/root_v6.08.06.source.tar.gz
$ cd /usr/local
$ sudo tar zxvf ~/root_v6.12.06.source.tar.gz
```

❶ ソースコードを落としてくる
❷ /usr/local に展開

※ sudo が使えなければ、システム管理者に
相談するか、他の場所に展開する

```
$ sudo mkdir root-6.12.06/cmake_build
$ cd root-6.12.06/cmake_build
$ sudo cmake ..
（省略）
-- Configuring done
-- Generating done
-- Build files have been written to: /usr/local/root-6.12.06/cmake_build
$
```

❸ CMake を実行

※ ここまで出れば CMake は成功

CMake でのビルド

```
$ sudo make -j 8                                    ❹ make を実行
（省略）
[100%] Linking CXX shared library ../../lib/libEve.so
[100%] Built target Eve
[oxon@oxon2014 cmake_build]$
```

※ ここまで出ればビルド成功

ちょっと遊んでみる

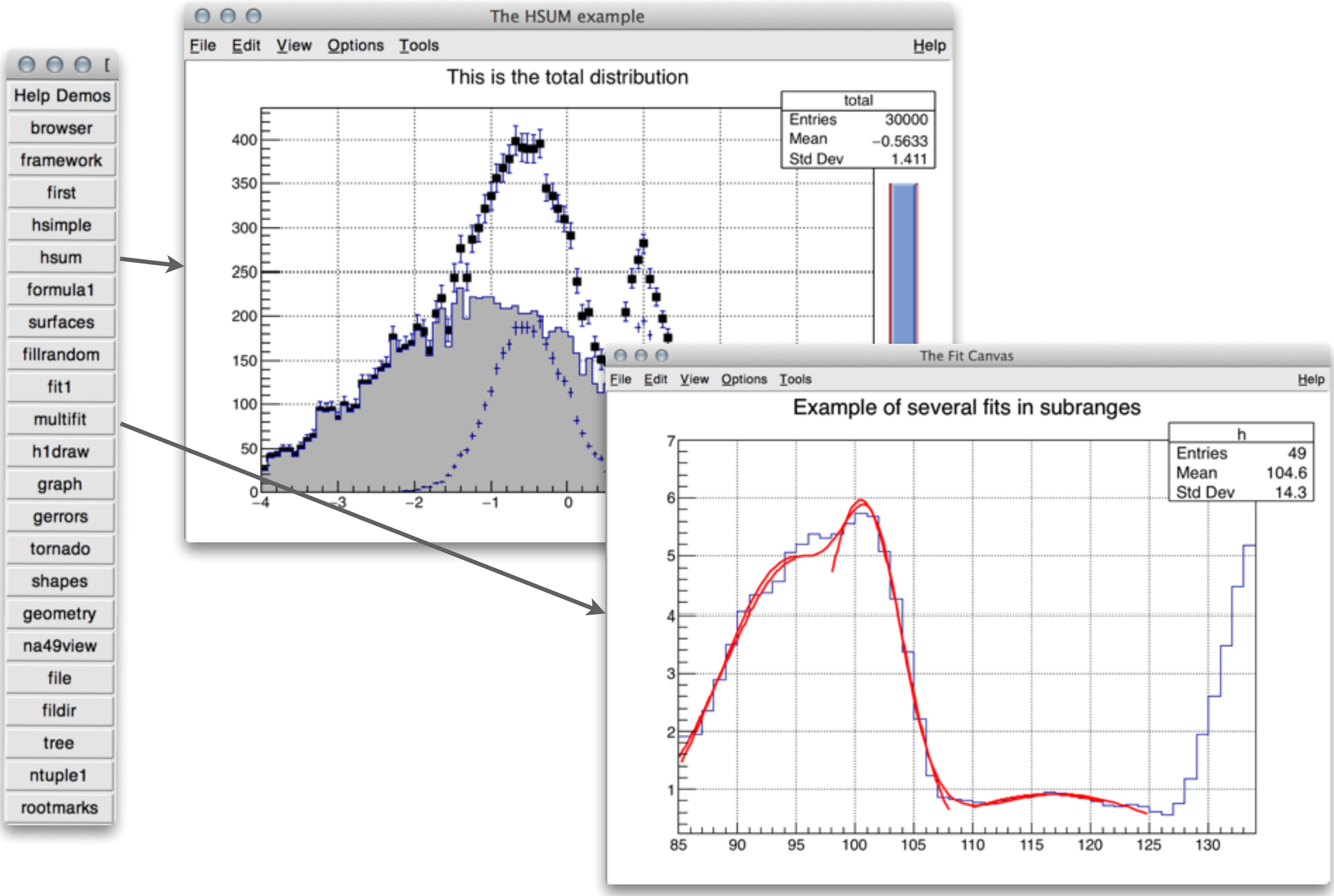
```
$ cd $ROOTSYS/tutorials/
$ root

Welcome to the ROOT tutorials

Type ".x demos.C" to get a toolbar from which to execute the demos

Type ".x demoshelp.C" to see the help window

==> Many tutorials use the file hsimple.root produced by hsimple.C
==> It is recommended to execute hsimple.C before any other script

root [0] .x demos.C
root [1]
```

❶ チュートリアル置き場に移動
❷ ROOT を起動

※ rootlogon.C が存在するので、表示メッセージが異なる

❸ demos.C というスクリプトを実行

デモの例


Help Demos
browser
framework
first
hsimple
hsum
formula1
surfaces
fillrandom
fit1
multifit
h1draw
graph
gerrors
tornado
shapes
geometry
na49view
file
fildir
tree
ntuple1
rootmarks
The HSUM example
File Edit View Options Tools Help
This is the total distribution
total
Entries 30000
Mean −0.5633
Std Dev 1.411
The Fit Canvas
File Edit View Options Tools Help
Example of several fits in subranges
h
Entries 49
Mean 104.6
Std Dev 14.3

質問の仕方

ソフトウェアの質問作法

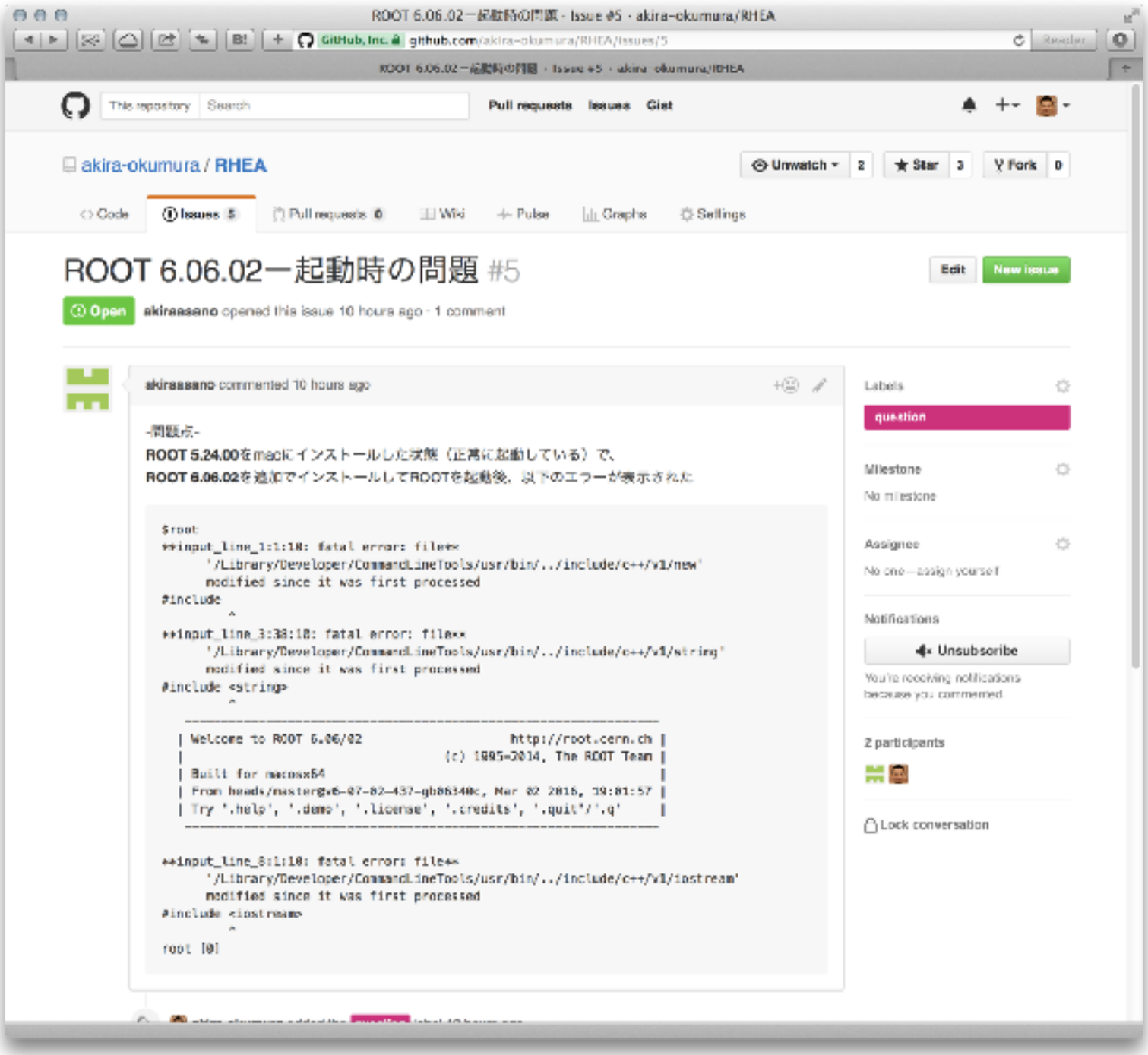
- ROOT や他のソフトウェアを研究で使い始めるとたくさんの疑問や問題にぶつかる
- 分からないものを悩んでも答えは出てこない
- 下手に悩むのは時間の無駄
- 素人のぶつかる問題の多くは、世界の誰かがとっくに経験している
- 質問せよ
  - 検索して解決する問題はググれ
  - まわりに聞け
  - ネットで聞け（後述）
  - それでも駄目なら自力で解決するか諦める

ググレカス

- 「人に質問する前にそれくらいググれ（Google で検索しろ）このカス野郎」という意味のネットスラング
- ありがちな例

  M1「ROOT 起動しようとしても動かないんですけど」
  怖い D3「なんかエラー出るの？」
  M1「command not found って出てます」
  D3「それをまず言えよ。英語の意味分かってる？」
  M1「いや、ちゃんと考えてませんでした。コマンドがないって意味ですか？」
  D3「ちゃんと分かってんじゃん。まず英語読めよ、カス」
  M1「で、なんで起動しないんですかね？意地悪しないで教えてくださいよ」
  D3「そのエラーでググレカス」 **※フレーズ検索（“”で囲む）がお勧め**
  M1「あ、なんか日本語の解説が出てきました。1. タイプミス 2. パスが通っていない…」
  D3「じゃあ順番にそれ読んで問題切り分けろ」
  M1「すみません、解決しました。$ROOTSYS/bin が PATH に入ってませんでした」
- 多くのソフトウェアの質問はググって解決します
- 日本語の検索結果ばっかり読まないこと
- この講習会では、ググらず即座に質問してもらって大丈夫です

ググレカスの例



- 意図的にググレカスな質問を載せてもらいました
  - https://github.com/akira-okumura/RHEA/issues/5
- エラーの１行目でググると、同じ問題と解決策が出てきます
  - https://root.cern.ch/phpBB3/viewtopic.php?t=19645

質問の仕方

- 質問をちゃんとする
  - たまに「動かない」とだけ言ってくる学生がいる
  - 「そうですか、動かないですか」としか返しようがない
  - 「お母さん、おしっこ」と同レベル
- ソフトウェア質問の大原則
  - 環境（OS、ROOT のバージョンなど）を伝える
  - 症状を伝える
  - エラーや出力を全て伝える（意訳しない、改変しない、省略しない）
  - 問題を再現する最小例を渡す（数百行あるバグったコードは誰も読んでくれない）
  - 試したことがあれば伝える（同じ手間を相手に取らせない）
- 「TV が映らない」とだけ言われて「電源ケーブル繋がってる？」からやり取りを始めたくない

駄目な質問例

「ROOT 起動しないんですよね、なんかエラー出てて」

- 質問は何？
- なんてエラー？
- どうやって起動しようとしたの？
- OS は？ROOT のバージョンは？
- どうやってインストールした？
- ターミナルになんて打ったの？
- ググった？

良い質問例（メールの場合）

ROOT コマンドを実行しても command not found と表示され起動できません。なにを見直せば良いでしょうか。

■ 環境
- OS X 10.9
- ROOT 6.06/02

■ インストール方法
奥村 PDF のやり方と同じく configure、make でビルドしたが、管理者権限がないので $HOME/root-6.06.02 でビルドした。

■ 症状
root コマンドが見つからない

$ root
bash: command not found: root

■ 試したこと

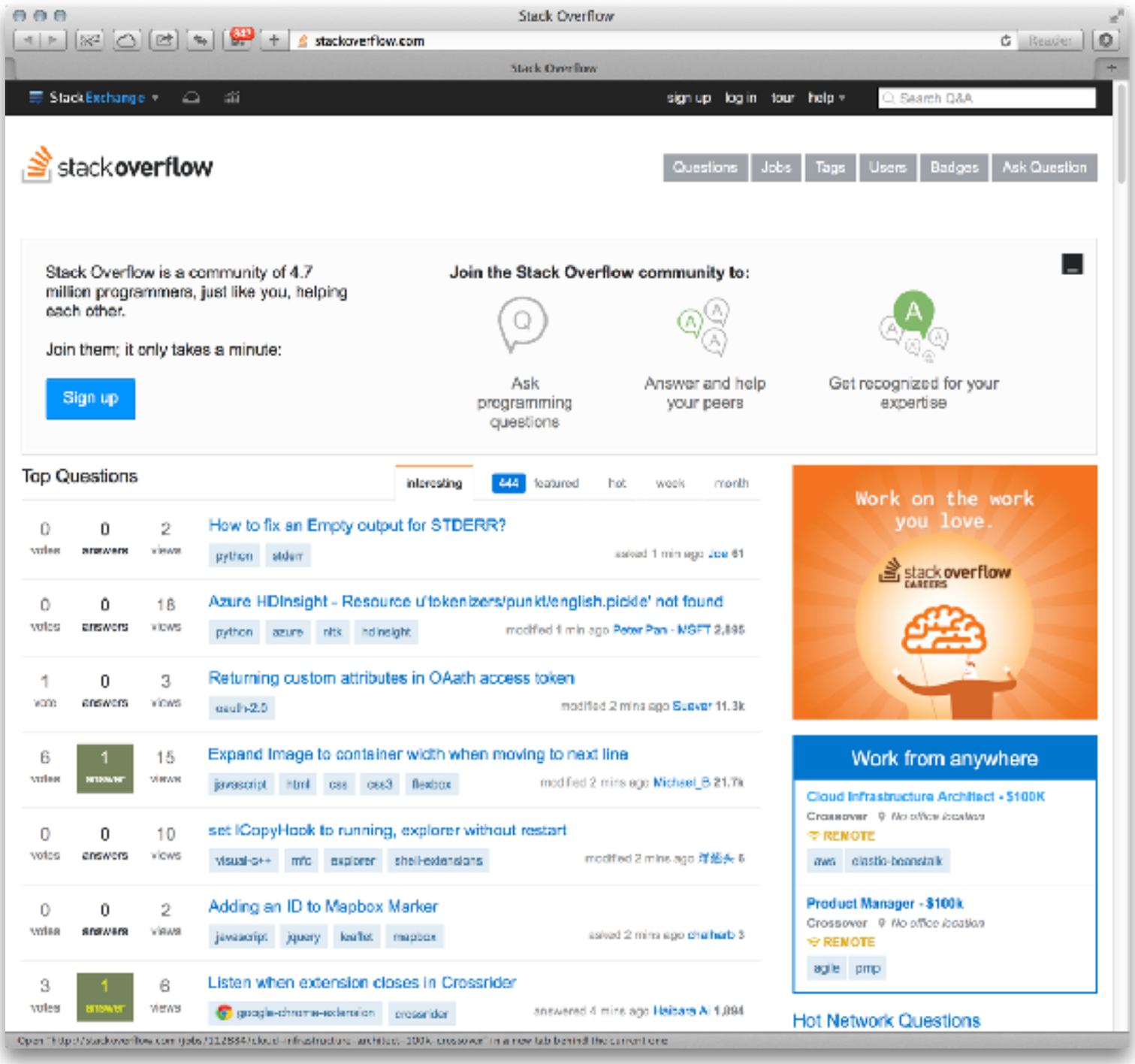
- $HOME/root-6.06.02/bin/root が存在するのは確認した
- which root を実行しても root not found と出る

ここまで書いてあれば「PATH が通っていないから echo $PATH をしろ、.bashrc を添付しろ」とすぐ返信できる

ネットで質問する（ROOT 関連の場合）

- ROOT の公式メーリングリストで質問する
  - https://groups.cern.ch/group/roottalk/default.aspx
  - 過去ログを読める
  - 参加するとメールが配信され、質問もできる（もちろん英語）
  - ググレカスとか言われない
  - 1–2 日で多くの場合は ROOT 開発者から返事が来る
  - すごい基本的なことまで質問しない
  - ROOT に直接関係のない C++ や Python の質問はしない
- ROOT の公式掲示板で質問する
  - https://root-forum.cern.ch
  - 返信がつかない場合がある
  - Google 検索で引っかかりやすい
  - 添付ファイルのサイズ制限が緩い

ネットで質問する（C++/Python 関連の場合）



- stackoverflow で聞く
  - http://stackoverflow.com/
  - 多種多様なプログラミング関係の質問を受け付けてくれる
  - 世界中の親切な人（暇な人）が答えてくれる
  - カスな質問をすると怒られることがある
  - ほとんどの基本的な質問は、既にネット上に回答が転がっていると思ったほうが良い

質問する上で大事なこと（ソフトウェアに限らず）

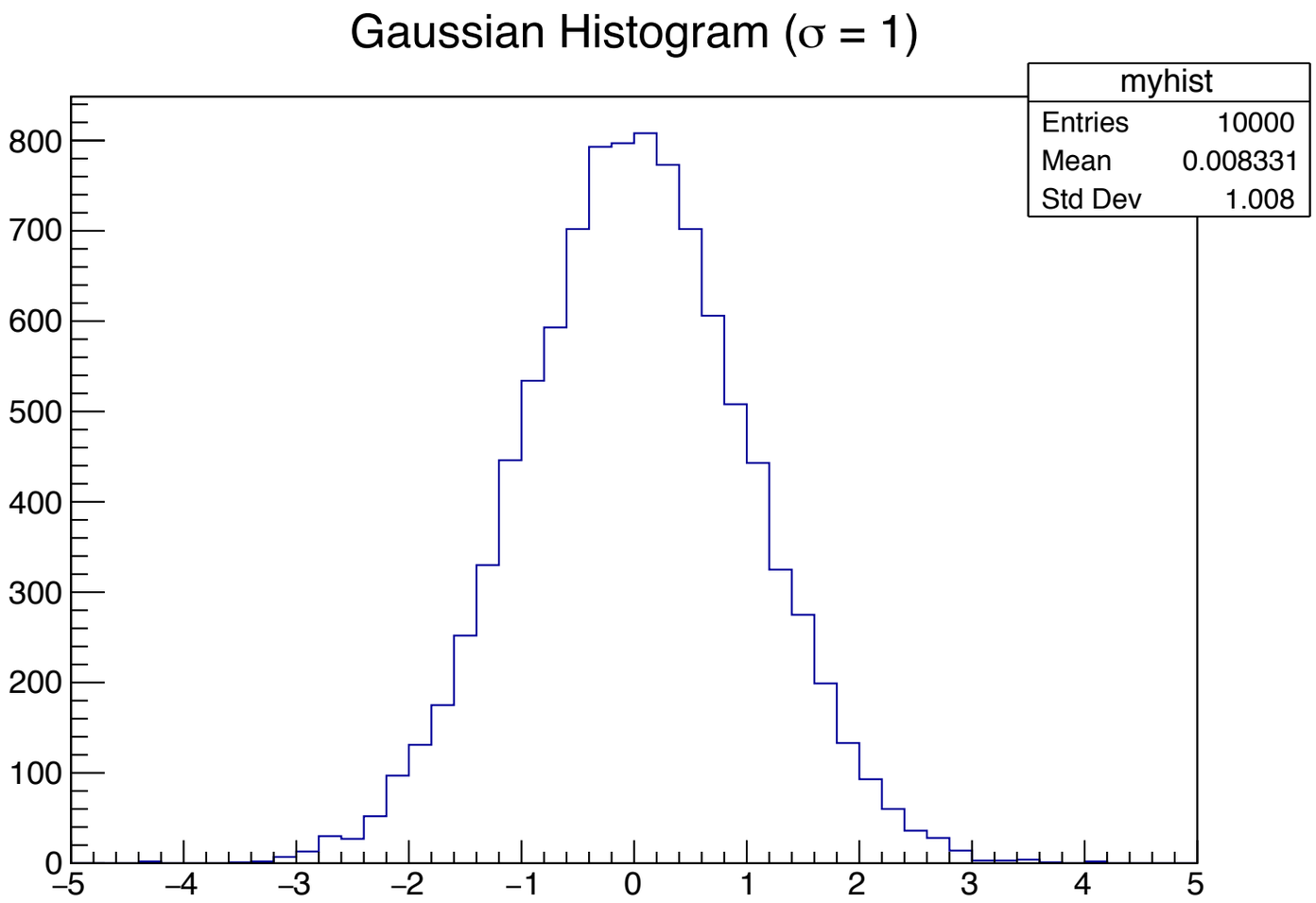
- 英語から逃げない
- 自分の問題を解決することだけを考えない
- 同じ問題を持つ誰かが未来に参照できるように心がける
- 質問した後に自己解決した場合は、その解決方法を自分から質問相手に伝える
- 相手から解決策を得たら、それで実際に解決したのかどうかの結果報告を忘れずにする
- 同時に複数投稿をしない（例えば ROOT のメーリングリストと掲示板）

ROOT の操作に慣れる

ROOT の使い方

- ROOT インタプリタに 1 行ずつ入力する方法
  - 動作確認や簡単な解析には良いが、すぐにやってられなくなる
- 複数行の C++ コードをファイルに保存し ROOT インタプリタから実行する
  - 使い回せる
  - エディタ上で編集するので楽
- 複数行の（同上）インタプリタからコンパイルして実行する
  - 処理速度が早い
  - C++ のコードとして不完全なものはコンパイルできない（ROOT 5）
- C++ アプリケーションから利用する（中級者向け）
- Python から使う（第 5 回くらいでやります）

正規分布のヒストグラム



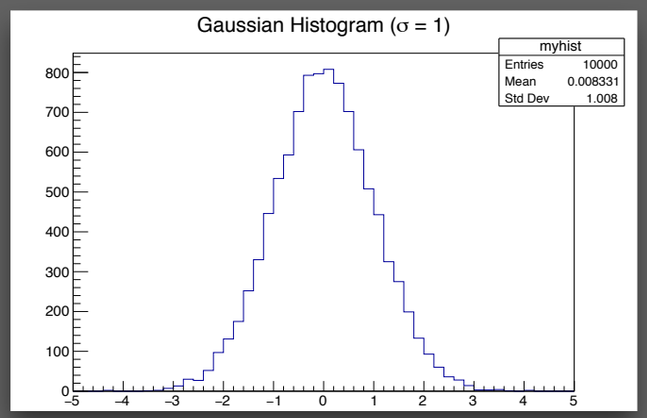

```
$ root
root [0] TH1D* hist = new TH1D("myhist", "Gaussian Histogram (#sigma = 1)", 50,
-5, 5)
root [1] hist->FillRandom("gaus", 10000)
root [2] hist->Draw()
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
root [3] c1->SaveAs(“hist.pdf”)
```

❶ 目的に応じてグラフやヒストグラムを作る
❷ データ解析の操作をする
❸ 描画する
❹ 図に保存する

✓ ROOT インタプリタは C++ を解釈できる
✓ ROOT プロンプトに 1 行ずつ入力し実行する

少し解説（新出用語は聞き流してください）

```
root [0] TH1D* hist = new TH1D("myhist", "Gaussian Histogram (#sigma = 1)", 50,
-5, 5)

root [1] hist->FillRandom("gaus", 10000)

root [2] hist->Draw()

Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1

root [3] c1->SaveAs(“hist.pdf”)
```



❶ TH1D という 1 次元ヒストグラム用クラスの
インスタンス（オブジェクト）を新たに作る
❷ コンストラクタの引数で、その仕様を決める

❸ オブジェクトに対して色々と操作をする。この場合は、
ガウス分布（正規分布）で乱数を 10,000 回詰める

❹ 多くの ROOT クラスは、Draw() というメソッド
（メンバ関数）を呼んで描画することができる

❺ 描画先（キャンバス）を作っていないので、デフォルト
の大きさのものが c1 という名前で自動的に作られる

❻ Draw した図は、色々な画像形式で保存可能。c1 も
ROOT のオブジェクトなので、メソッドを多く持つ。
基本的に PDF で保存すること（EPS は今どき使わない）。

1 行ずつ入力するのは面倒くさいので関数にする

```
$ cat first_script.C
```

❶ スクリプトとして、ファイルに書いてしまう。.C という拡張子を使うのが ROOT では一般的

```
void first_script()
{
  TH1D* hist = new TH1D("myhist", "Gaussian Histogram (#sigma = 1)", 50, -5, 5);
  hist->FillRandom("gaus", 10000);
  hist->Draw();
}

$ root
root [0] .x first_script.C
```

❷ ファイル名と同じ関数にすること

❸ .x という ROOT の固有コマンドを使って実行する

参考書に掲載されているプログラムは Git で入手可能

```
$ git clone https://github.com/akira-okumura/RHEA.git
$ cd RHEA
$ ls
Makefile  README.md RHEA.tex  fig       src       tex
$ cd src
$ ls first_script*
first_script.C   first_script2.C  first_script2.py first_script3.C
```

関数に引数を持たせて汎用性を高める

```
$ cat first_script2.C
```

❶ 引数を持った関数を新たなファイルに作る

```
void first_script2(int nbins, int nevents)
{
  TH1D* hist = new TH1D("myhist", "Gaussian Histogram (#sigma = 1)", nbins, -5,
5);
  hist->FillRandom("gaus", nevents);
  hist->Draw();
}

$ root
root [0] .x first_script2.C(500, 100000)
```

❷ 実行時に引数を渡すことができる

コンパイルして実行する

```
$ cat first_script3.C
#include "TH1D.h"

void first_script3(int nbins, int nevents)
{
  TH1D* hist = new TH1D("myhist", "Gaussian Histogram (#sigma = 1)", nbins, -5,
5);
  hist->FillRandom("gaus", nevents);
  hist->Draw();
}

$ root
root [0] .x first_script3.C+(500, 1000000)
Info in <TMacOSXSystem::ACLiC>: creating shared library /Users/oxon/git/RHEA/
src/./first_script3_C.so

Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

❶ コンパイルしても動くように、ヘッダーファイルを
ちゃんとインクルードする

❷ コンパイルするときはファイル名の後ろに + をつける。
複雑なプログラムの場合、処理速度が向上する。

❸ コンパイル済みの共有ライブラリ（.so）が生成される

Python から実行する

```
$ cat first_script2.py

import ROOT                                    ❶ $ROOTSYS/lib/ROOT.py をインポートする

def first_script2(nbins, nevents):  ❷ C++ と構造としては一緒だが、文法が色々と違う
    global hist
    hist = ROOT.TH1D('myhist', 'Gaussian Histogram (#sigma = 1)', nbins, -5, 5)
    hist.FillRandom('gaus', nevents)
    hist.Draw()

$ ipython
Python 3.6.5 (default, Mar 30 2018, 06:41:53)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.2.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import first_script2          ❸ 自分のスクリプトをインポートする

In [2]: first_script2.first_script2(500, 100000)   ❹ 関数を呼び出す
Info in <TCanvas::MakeDefCanvas>:  created default TCanvas with name c1
```

※ 他にもやり方はあります

タブ補完機能の使い方

```
$ root
root [0] TH1D* hist = new TH1D("myhist", "Gaussian Histogram (#sigma=1)", 50,
-5, 5)
root [1] hist->FillRandom("gaus", 10000)
root [2] h
```

❶ ここでタブキーを打つと…

```
h
(省略)
hort>
host2net
root [2] hi
root [2] hist
root [2] hist->

AbstractMethod
Add
AddAt
AddBinContent
(省略)
```

❷ 候補が表示され…
❸ hi まで入力しさらにタブを打つと hist と補完される

❹ メソッド候補を出すには hist-> まで入力しタブ

履歴検索とカーソルの移動

- Control + R（C-r とか Ctrl-r とか書きます）で過去の入力を検索できる
  - いちいちコピペしない
  - いちいち入力しない
- カーソル移動
  - 上下左右 C-p (previous)、C-n (next)、C-f (forward)、C-b (backward)
  - 行頭と行末 C-a (ahead)、C-e (end)
  - 矢印キーは使わない
  - 常に人差し指を F と J のキーに置く
  - A キーの左が Caps Lock の場合、Control に置き換える設定をする
- 他にもいくつかある（Emacs や shell と同じ）
  - C-h、C-d、C-k、C-y、C-t あたりは便利
  - Delete も Backspace キーも使わない

ROOT の公式ドキュメント

- 『ROOT User's Guide』
  - https://root.cern.ch/root-user-guides-and-manuals
  - HTML 版か PDF 版（642 ページ！）
  - 膨大だけど最も詳しい公式解説書
  - 単語で検索をかけて飛ばし読みでも十分
- Reference Documentation
  - https://root.cern.ch/doc/master/index.html
  - C++ のソースコードから自動生成された HTML
  - 例えばヒストグラムのクラスが何をしているか理解するには https://root.cern.ch/doc/master/classTH1.html を読む

第 1 回のまとめ

- GitHub の理解
- 参考書 PDF の生成の仕方
- ROOT のインストール
- ROOT の動作確認
- 基本的な使い方とスクリプトの実行の仕方
- 質問の仕方

- 分からなかった箇所は、各自おさらいしてください