Introduction to High-Energy Physics Analysis

Didelių energijų fizikos eksperimentinių duomenų analizės pagrindai

Homework Problem Set 1 (A&B)

2024-02-22

Homework

Due dates:
Problem Set 1A: 2024-03-04 23:59 (EET);
Problem Set 1B: 2024-03-11 23:59 (EET).

General guidelines:

1. Problem Set 1A: Problems 1–4;
2. Problem Set 1B: Problems 5–7.
3. Problems 2–6 should be submitted via gitlab as initiated via Problem 1.
4. Each Problem should have its own code file.
5. Name code files using the following format: problemX.Y, where X is the Problem number and Y is an appropriate extension.

Problems:

1. **git**: Create and share the gitlab project with an instructor, where the homework scripts are uploaded.
2. **bash**: Create a bash script that understands user-specified arguments.
   Arguments: *fancy* and *pants* should be recognized.
   Let *fancy* print "Oh" to the terminal, *pants* print "yes!".
   The script should be used as: <script name> <arguments>, i.e.,
   > $ bash problem2.sh fancy
   > Oh
   Note that "argument processing" is a synonym expression to "argument parsing".
3. **bash**: Create a bash script to checkout (branch off) from a user-selected commit.
   It should contain the following arguments:
   (a) *date*: checks out a commit given a commit date;
   (b) Commit *number*: checks out a commit given a commit number, i.e., chronologically counting from the first commit;
   (c) *help*: prints a "standard" help (args. and descriptions).

   The script should be used as: <script name> <arguments>.
   Use a local repo for git manipulations.
4. **bash**: Write a new script that extends the script from the previous problem with the following arguments:
   (a) *new-branch*: creates a new specified branch;
   (b) *del-branch*: deletes a specified branch;

(c) *sel-branch*: switches to a specified branch.

In addition, implement argument parsing with shorthand arguments, i.e., '-a' or '-a <value>'. Usage example:
$>$ $ bash problem4.sh -d old_branch_name

5. **bash**: Create a command-line tool (script) that compares two directories. In particular, the script should have the following features:

   (a) finds and prints the differences in directories, i.e., between the directory trees (something like: "dir A has A/B/c.file");

   (b) has a deep comparison functionality if argument *deep* is specified. Use sha256sum to achieve that.

   Think: structure comparison only or structure&content (data) comparison.

6. **bash**: Create a script that generates $\sim$2 MB worth of data in the following format: comma separated values (CSV format) of 8 numbers per line. The script should have the following features:

   (a) Saves the output to a file 'vector-8.csv'.

   (b) Numbers must be selected **randomly**, i.e., all $8 \times n_{\text{lines}}$ numbers must* differ.

   (c) Use a smart way to let the script keep aware of the growing size.

   * There is always a tiny probability (= an unintentional artifact of a random number generator) to find two or esp. more identical numbers.

7. **bash**: Create a new script that 'digests' the CSV with the following features:

   (a) prints an "invariant mass" for each line (two-body system), where the first four numbers are the first 4-momenta and the next four — the second 4-momenta.

   (b) Output (AKA stdout) should be redirected to a file.

   Let's call the line in the CSV "an event".