Corso Abilità Informatiche e Telematiche 2024/2025

Exam

Teacher: Dr. Milena Valentini, milena.valentini@units.it
Teacher: Dr. David Goz, david.goz@units.it

1 Exam instructions

- Choose and solve four out of the eight Bash exercises proposed;
- in addition, solve the Python exercise proposed;
- create a repository on `GitHub` (https://github.com/) containing the scripts of the solved problems. The `README.md` file associated with the GitHub repository is meant to tell the teachers about your exam, i.e.: i) the list of exercises you have addressed, ii) how to use your scripts, iii) the output that your scripts will produce and where it will be displayed and saved. Create a directory for each exercise which contains the script(s) that perform(s) the task(s) required by the problem and the additional files if necessary. The teachers should be able to test your script(s) by simply entering the associated directory and following your instructions;
- send an email to the two teachers attaching the link to the GitHub repository. The teachers should be able to clone your repository smoothly; they will try to execute your scripts by following your instructions, and then have a look at the scripts themselves;
- the exam material must be posted to the teachers at least one week before the examination date;
- the candidate must register online for the examination date.

2 Exercises

Problem 1: Bash scripting

Write a shell script that:

- through a `for loop` creates a bunch of dummy files within the working directory;
- creates an array containing all the files (not directories) in the current working directory;
- checks if the list is empty, otherwise it displays on the standard output all the files in the current directory;
- it renames all files to begin with today's date in the following format: YYYY-MM-DD. For example, if a picture of my dog was in the current directory and today was December 10, 2024 it would change the name from *my_dog.jpg* to *2024-10-12-my_dog.jpg*;
- Hint: `touch`, and `date` commands might help. The latest prints the system date and time. Look through the documentation to set the correct format;
- Suggestion: the script's comments and documentation are highly recommended.

Problem 2: Bash scripting

Write a shell script to check and see if:

- the file `/etc/shadow` exists;
- if it does exist;
  - display "*Shadow passwords are enabled*" in the standard output;
  - show the file content in the standard output;
- if it does not exist return a non-zero exit status;
- next, check if you have the permission to write in the file;
  - if you can, display "*You have permission to edit* `/etc/shadow`". Return a zero exit status;
  - if you cannot, display "*You do NOT have permissions to edit* `/etc/shadow`". Return a non-zero exit status.
- Hint: `$ help test` command might help!
- Suggestion: the script's comments and documentation are highly recommended.

Problem 3: Bash scripting

Write a shell script that:

- consists of a function that displays the number of files (not directories!) in the current directory;
- name this function "*file_count*" and call it within the bash script;

- if you use a variable in your function, remember to make it a local variable;
- constraint: the function can only use the `ls`, `grep`, and `wc` commands to accomplish the task;
- Hint: a pipe among `ls, grep, wc` might help. Test the correctness of the implementation by creating a bunch of dummy files within the working directory using the `touch` command;
- Suggestion: the script's comments and documentation are highly recommended.

Problem 4: Bash scripting

Modify the script from the previous exercise.

- make the "*file_count*" function accept a directory as an argument;
- the function tests if the passed argument is a regular directory, otherwise it handles the error providing a meaningful message, and returns a non-zero exit status;
- the function displays the name of the directory followed by a colon;
- finally, the function displays the number of files in the standard output on the next line;
- call the function three times: first on the "*/etc*", next on the "*/var*" directory and finally on the "*/usr/bin*" directory.
- Example function output:
  ```
  /etc:
  110 files
  ```
- Suggestion: the script's comments and documentation are highly recommended.

Problem 5: Bash scripting

Write a shell script that:

- creates an empty file;
- writes on that file a column containing the first ten integer numbers (from 1 to 10) using a `for` or `while loop`;
- it evaluates the total sum of the integer numbers through the `awk` command (or the corresponding GNU implementation `gawk`) and prints the total sum on the standard output;
- Hint. The result can be checked using the Gauss summation formula;
- Suggestion: the script's comments and documentation are highly recommended.

Problem 6: Bash scripting

Write a shell script that:

- writes a file with the following content:

```
# control of memory requirements
BoundaryLayerFactor      3.0
MaxMem                   512
MaxMemPerParticle        240
PredPeakFactor           0.8
```

- it changes in place (i.e. changes must be saved in the same file) the value of *MaxMem* from 512 to 1024 using the `awk` command (or the corresponding GNU implementation `gawk`);
- Suggestion: the script's comments and documentation are highly recommended.

Problem 7: Bash scripting

Write a shell script that:

- displays on the standard output and saves on file *lscpu.txt* information about the CPU architecture using the `lscpu` command;
- writes a new file with the following format and content:

```
CPU architecture:
      Cpu name      :  ${CPU}
      Threads       :  ${THREADS}
      Cores         :  ${CORES}
      NUMAs         :  ${NUMA}
```

  where `CPU, THREADS, CORES, NUMA` are the corresponding values extracted from the `lscpu` command of *"Model name", "Thread(s) per core", "Core(s) per socket", "NUMA node(s)"* respectively;
- Suggestion: the script's comments and documentation are highly recommended.

Problem 8: Bash scripting

Write a shell script that:

- accepts a directory as an argument;
- checks that the directory exists, otherwise it handles the error providing a meaningful message, and returns a non-zero exit status;
- enters the directory;
- lists just all the regular files;

- lists just the files that are directories;
- lists just the empty files;
- Suggestion: the script's comments and documentation are highly recommended.

Problem 9: Python scripting

Download the following file: click here to access the file to analyse its content.
The file has been produced by post-processing a cosmological hydrodynamical simulation of structure formation, at redshift $z = 6$.
The header of the file describes its content and tells you the units of measure of the quantities in each of the file columns. Each line contains properties of a halo or cosmological structure, which features multiple components: black holes, stars, dark matter (DM), gas.
Write one (or more, if needed) Python routine(s) that:

- plot the DM mass of each halo as a function of its baryonic mass (i.e. the sum of the stellar and the gas mass); try to overplot a linear fit, and comment.
- Focus on the most massive structure (by just selecting it from the first column with e.g. the numpy.where() instruction); compute the distance of each of the other structures from the most massive one, and plot results in the plane y=total mass VS x=distance; make tests to understand which combinations of axis scales (e.g., linear-linear, log-linear, log-log, ...) allows you to better appreciate the distribution.
- Plot the histogram that shows the DM mass distribution of the haloes; overplot mean and median of the distribution as vertical lines; remember to always put a legend and to write on the plot relevant information (e.g., values of the mean and median).
- Plot the projected distribution of all the haloes in the file in the planes x-y (left-hand panel) and z-y (right-hand panel). Assign each halo (point in the scatter plot) a colour which encodes its gas mass, and a size which scales with its stellar mass. Try to add a third panel (at the bottom of one of the former two) by including the x-z projection.
- Plot the BH mass (y-axis) VS stellar mass (x-axis) relation, for all those haloes having a BH mass larger than 8×10^5 $\mathrm{M}_\odot$/h; try to overplot a linear fit, and comment.
- Produce a cumulative 2-D histogram providing the number of haloes per bin of mass (x axis) and distance from each of the other haloes (y axis) for those haloes whose mass is larger than 3.07×10^9 $\mathrm{M}_\odot$/h. The final (cumulative) histogram will be obtained by summing up the histograms of each of the haloes (5 in total) more massive than 3.07×10^9 $\mathrm{M}_\odot$/h. The colour in each bin will encode the number of haloes in the mass-distance bin. (Hint: adopt the same mass and distance bins in all the histograms, i.e. in each histogram for each of the 5 haloes).

Comment: The Python script has to be properly commented and documented. Figures produced have to be publication-quality: make sure all the axis have the proper label, that there is the legend, that quantities are fully readable (i.e. evaluate whether a linear or a log scale fits better your need for each plot, ...)