

UCSB, Physics 129AL, Computational Physics: Problem Set 1

Zihang Wang (UCSB), zihangwang@ucsb.edu

Due: Jan 21, 11:59PM

Problem Set Submission Guideline

A) Github Submissions We will use GitHub for problem set submissions. To access the problem set, please **fork** and **clone** the **forked** repository to your local virtual machine. **Please complete the problem set in this forked directory.** Submit a **pull request** for merging into the main branch before the problem set due date.

B) Export Command History: Please submit all of your commands (including errors) in a text file named “history.txt” in the problem set directory. For instance, to select and append the last 10 commands to the file, use the following history command: “history | tail -n 10 >> history.txt”. Note that the last command in the text file will be the history command mentioned above.

C) .tar.gz File compression and submission on Github for each problem set, you are asked to submit the compress version of the problem set to github via git operation. Here is a step by step guideline,

1. Use the **tar** command to compress the problem set directory into a **single** “.tar.gz” file.
2. Obtain the sha256sum by running “sha256sum #.tar.gz” (#: problem set number).
3. Echo the **full sha256sum** to a text file named “sha25sum_problem_set.txt”.
4. Initialize a git repository named “Archive_P#” (#: problem set number) on your local machine, and move both the “tar.gz” file and the “sha25sum_problem_set.txt” file to the repository.
5. Create an empty **public** directory under the **same name in your own github account**.
6. **Push** this local repository to the remote repository.

In summary, by the end of the due date, you should have two directories: one is the **forked directory, where you submit your pull request**, and the

other is **your own archive directory**. In this problem set, you are asked to follow the above guideline, otherwise, you will receive zero for the problem set.

I put some notes for references, and the problem set starts with “Problem #”. **Remember, talk to your fellow students and work together. You will find it much easier than working alone. Good luck!**

1 Basic Bash command usages

First created in Bell Labs in the late 1960s and early 1970s, UNIX treats everything as a file where it can be managed in a unified manner. It offers a rich set of commandline tools for performing file operations. We will look at a few basics commands in Linux here. Fig.1: we explore few basic commands in Linux file management, e.g. “mkdir”, “ls”, “cd”, “echo”, “cat”, “*l,ll*”. Fig.2: “ls -l” command lists all files and their properties. The total is usually used to estimate the disk usage of a given directory. Fig.3: Standard input, output, and error. Fig.4: Grep keywords in a file. Fig.5: Find files and execute commands. Fig.6: Pipelines. Fig.7: General commands, e.g. “touch”, “mv”, “rm”, “cp”, “chmod”, “ln -s”, “alias”, and “unalias”. Fig.8: Archives and Tar format.

The screenshot shows a terminal window with a dark background and light-colored text. At the top, there are several tabs with the first one labeled "zhang@zhang-VirtualBox: ~/Desktop/P129_data/lecture1...". Below the tabs, there are four red arrows pointing to specific parts of the terminal interface:

- A red arrow points to the first tab with the text "Change to the directory".
- A red arrow points to the second tab with the text "List all files".
- A red arrow points to the third tab with the text "Making a directory".
- A red arrow points to the fourth tab with the text "File name".

The terminal window displays the following command history:

```
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1$ mkdir lecture1_data
zhang@zhang-VirtualBox:~/Desktop/P129_data$ ls
lecture1_data      lecture1_material.txt  ps_link
lecture1_material.sh  physics129_Data1.sh  test.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data$ cd lecture1_data/
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ echo hello world!
hello world!
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ echo hello world! > hello_world.txt
Display the contents of a file
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ cat hello_world.txt
hello world!
Append text to an existing file
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ echo a second hello
to the world! >> hello_world.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ cat hello_world.txt
hello world!
a second hello to the world! Overwrite contents in an existing file
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ echo a new hello to
the world! > hello_world.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ cat hello_world.txt
a new hello to the world!
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$
```

Figure 1: basic commands.

```

zhang@zhang-VirtualBox:~/Desktop/P129_data$ ls -l
total 12
-rw-rw-r-- 1 zhang zhang 69 Sep  5 20:42 lecture1_material.sh
-rw-rw-r-- 1 zhang zhang 60 Sep  5 20:39 lecture1_material.txt
-rwxrwxr-x 1 zhang zhang 193 Sep 18 13:03 physics129_Data1.sh
lrwxrwxrwx 1 zhang zhang 28 Sep 18 12:05 ps_link -> ../../P129_ps/physics129_PS1
.zsh
zhang@zhang-VirtualBox:~/Desktop/P129_data$
```

Total number of 1KB blocks (min. 4)

List full file information

File name

Symbolic link name

Linked location

Time/Date created or the most recent update

File size (bytes)

Groups

Owner

Types and permissions (-rwxrwxr-x)

First: file type

Others (r-x)

Figure 2: "ls -l" command.

Direct the output and error of
"ls" command to two files

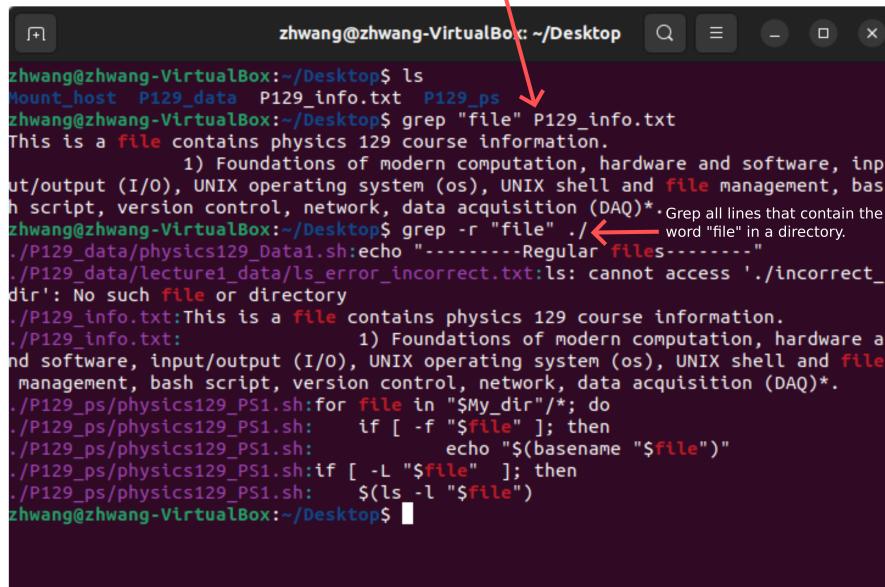
```

zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1...
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ ls ./ > ls_out.txt 2>
> ls_error.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ ls hello_world.txt ls_error.txt ls_out.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ cat ls_out.txt
hello_world.txt
ls_error.txt      Contains the standard output of the command "ls"           No error (empty)
ls_out.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ cat ls_error.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ ls ./incorrect_dir >
ls_out_incorrect.txt 2>ls_error_incorrect.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ ls
hello_world.txt      ls_error.txt      ls_out.txt
ls_error_incorrect.txt  ls_out_incorrect.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ cat ls_out_incorrect
.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ cat ls_error_incorr
ct.txt
ls: cannot access './incorrect_dir': No such file or directory
zhang@zhang-VirtualBox:~/Desktop/P129_data/lecture1_data$ Error message

```

Figure 3: Standard input, output and error.

Grep all lines that contain the word, file



A screenshot of a terminal window titled "zhwang@zhwang-VirtualBox: ~/Desktop". The terminal shows several commands and their outputs:

```
zhwang@zhwang-VirtualBox:~/Desktop$ ls
Mount_host P129_data P129_info.txt P129_ps
zhwang@zhwang-VirtualBox:~/Desktop$ grep "file" P129_info.txt
This is a file contains physics 129 course information.
    1) Foundations of modern computation, hardware and software, input/output (I/O), UNIX operating system (os), UNIX shell and file management, bash script, version control, network, data acquisition (DAQ)*.
./P129_ps/physics129_PS1.sh:grep -r "file" ./          ← word "file" in a directory.
./P129_data/physics129_Data1.sh:echo "-----Regular files-----"
./P129_data/lecture1_data/ls_error_incorrect.txt:ls: cannot access './incorrect_dir': No such file or directory
./P129_info.txt:This is a file contains physics 129 course information.
./P129_info.txt:                                1) Foundations of modern computation, hardware and software, input/output (I/O), UNIX operating system (os), UNIX shell and file management, bash script, version control, network, data acquisition (DAQ)*.
./P129_ps/physics129_PS1.sh:for file in "$My_dir"/*; do
./P129_ps/physics129_PS1.sh:    if [ -f "$file" ]; then
./P129_ps/physics129_PS1.sh:        echo "$(basename "$file")"
./P129_ps/physics129_PS1.sh:if [ -L "$file" ]; then
./P129_ps/physics129_PS1.sh:    $(ls -l "$file")
zhwang@zhwang-VirtualBox:~/Desktop$
```

Figure 4: Grep keywords.

-rl option search all files in a directory
that contains the word "file", and output
the file names.

```

zhang@zhang-VirtualBox:~/Desktop$ ls
Mount_host P129_data P129_info.txt P129_ps
zhang@zhang-VirtualBox:~/Desktop$ grep -rl "file" ./
./P129_data/physics129_Data1.sh
./P129_data/lecture1_data/ls_error_incorrect.txt
./P129_info.txt
./P129_ps/physics129_PS1.sh
zhang@zhang-VirtualBox:~/Desktop$ find ./ -type f -name "*.txt"
./P129_data/test.txt
./P129_data/lecture1_data/hello_world.txt
./P129_data/lecture1_data/ls_error.txt
./P129_data/lecture1_data/ls_out_incorrect.txt
./P129_data/lecture1_data/ls_out.txt
./P129_data/lecture1_data/ls_error_incorrect.txt
./P129_data/lecture1_material.txt
./P129_info.txt
./P129_ps/problemset_info.txt
./P129_ps/problemset_data.txt
zhang@zhang-VirtualBox:~/Desktop$ find ./ -type f -name "*.txt" -exec grep -l
"file" {} +
./P129_data/lecture1_data/ls_error_incorrect.txt
./P129_info.txt
zhang@zhang-VirtualBox:~/Desktop$
```

find all files in the current directory that meet the following conditions.
 1) the type must be file.
 2) the file must contain "*.txt", where * means "anything".

we execute the command grep -l "file" {} when all the above conditions are met. The syntax {} is a placeholder that gets replaced by the output of find. The + means appending all files before executing grep.

Figure 5: "find" command.

```
zhang@zhang-VirtualBox:~/Desktop$ ls
Mount_host P129_data P129_info.txt P129_ps
zhang@zhang-VirtualBox:~/Desktop$ find ./ -type f -name "*.txt"
./P129_data/test.txt
./P129_data/lecture1_data/hello_world.txt
./P129_data/lecture1_data/ls_error.txt
./P129_data/lecture1_data/ls_out_incorrect.txt
./P129_data/lecture1_data/ls_out.txt
./P129_data/lecture1_data/ls_error_incorrect.txt
./P129_data/lecture1_material.txt
./P129_info.txt
./P129_ps/problemset_info.txt
./P129_ps/problemset_data.txt
zhang@zhang-VirtualBox:~/Desktop$ find ./ -type f -name "*.txt" | xargs grep
-l "file"
./P129_data/lecture1_data/ls_error_incorrect.txt
./P129_info.txt
zhang@zhang-VirtualBox:~/Desktop$ find ./ -type f -name "*.txt" | xargs grep
-l "file" | xargs grep -l "course"
./P129_info.txt
zhang@zhang-VirtualBox:~/Desktop$
```

The first command are executed, then the output is passed as the input of the second command via xargs

Multiple commands can be ordered via the pipeline (data flow).

Figure 6: Pipelines.

```
zhwang@zhwang-VirtualBox:~/Desktop$ ls
Mount_host P129_data P129_info.txt P129_ps          Create an empty file.
zhwang@zhwang-VirtualBox:~/Desktop$ touch empty_file.txt      Change (or move)
a file.
zhwang@zhwang-VirtualBox:~/Desktop$ ls
empty_file.txt Mount_host P129_data P129_info.txt P129_ps
zhwang@zhwang-VirtualBox:~/Desktop$ mv empty_file.txt new_empty_file.txt
zhwang@zhwang-VirtualBox:~/Desktop$ ls
Mount_host new_empty_file.txt P129_data P129_info.txt P129_ps          Remove a file
zhwang@zhwang-VirtualBox:~/Desktop$ rm new_empty_file.txt
zhwang@zhwang-VirtualBox:~/Desktop$ cp P129_info.txt Copy_P129_info.txt
zhwang@zhwang-VirtualBox:~/Desktop$ ls
Copy_P129_info.txt Mount_host P129_data P129_info.txt P129_ps          Copy a file
zhwang@zhwang-VirtualBox:~/Desktop$ chmod +x Copy_P129_info.txt      Change file
permission
zhwang@zhwang-VirtualBox:~/Desktop$ ls
Copy_P129_info.txt Mount_host P129_data P129_info.txt P129_ps
zhwang@zhwang-VirtualBox:~/Desktop$ ln -s ./P129_data/lecture1_material.txt lecture_material
zhwang@zhwang-VirtualBox:~/Desktop$ ls
Copy_P129_info.txt Mount_host P129_info.txt
lecture_material P129_data P129_ps          Create a symbolic link
zhwang@zhwang-VirtualBox:~/Desktop$ alias ll="ls -l"
zhwang@zhwang-VirtualBox:~/Desktop$ unalias ll
zhwang@zhwang-VirtualBox:~/Desktop$ ll
ll: command not found
zhwang@zhwang-VirtualBox:~/Desktop$
```

Figure 7: common Linux commands.

```

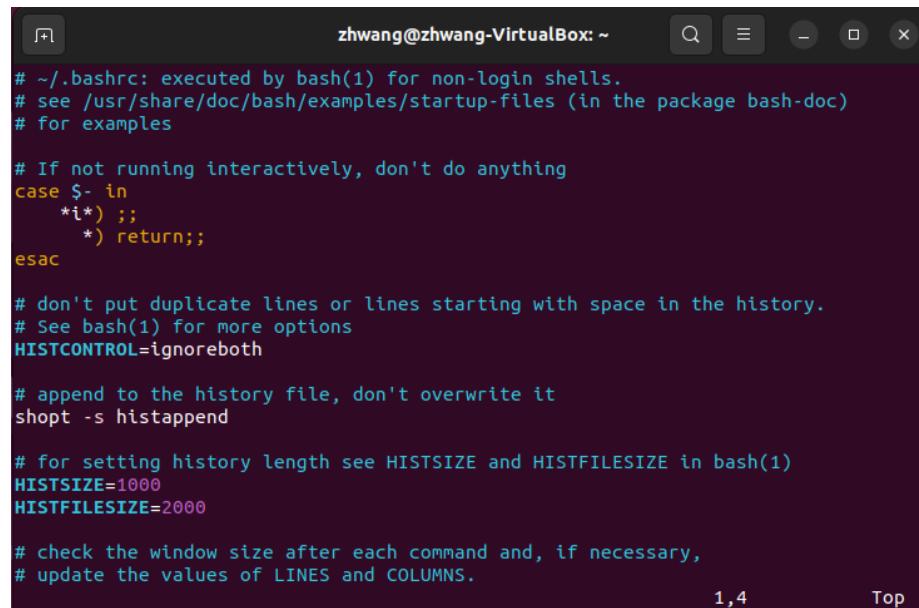
zhang@zhang-VirtualBox:~/Desktop$ ls
Copy_P129_info.txt Mount_host P129_info.txt
lecture_material P129_data P129_ps
zhang@zhang-VirtualBox:~/Desktop$ tar -czvf P129_ps.tar.gz ./P129_ps
./P129_ps/
./P129_ps/problemset_info.txt      c: create
./P129_ps/physics129_PS1.sh       z: gz format
./P129_ps/problemset_data.txt     f: file name
zhang@zhang-VirtualBox:~/Desktop$ ls
Copy_P129_info.txt Mount_host P129_info.txt P129_ps.tar.gz
lecture_material P129_data P129_ps           v: verbose
zhang@zhang-VirtualBox:~/Desktop$ mv P129_ps.tar.gz ./P129_data/
zhang@zhang-VirtualBox:~/Desktop$ cd P129_data/
zhang@zhang-VirtualBox:~/Desktop/P129_data$ ls
lecture1_data lecture1_material.txt physics129_Data1.sh test.txt
lecture1_material.sh P129_ps.tar.gz ps_link
zhang@zhang-VirtualBox:~/Desktop/P129_data$ tar -xzvf P129_ps.tar.gz
./P129_ps/
./P129_ps/problemset_info.txt      x: extract
./P129_ps/physics129_PS1.sh
./P129_ps/problemset_data.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data$ ls
lecture1_data lecture1_material.txt P129_ps.tar.gz ps_link
lecture1_material.sh P129_ps physics129_Data1.sh test.txt
zhang@zhang-VirtualBox:~/Desktop/P129_data$

```

Figure 8: tar command.

2 bash script usages

A Bash script is a text file that contains a series of commands discussed above. It provides a great way to automate tasks and perform various operations in a Linux or Unix environment. We will explore the basic functionality of Bash scripting. A bash script usually has ".sh" file extension that can be executed. It has a shebang line that indicates the bash in the /bin directory will be used, `#!/bin/bash`. We usually find the hidden file in the user home directory, named `~/.bashrc`, useful to customize bash shell based on our needs when we open the shell, e.g. alias or other script programs. We should note that it only applies to the local user, and if you want to apply system-wide changes, you should change `/etc/bash.bashrc`.



The image shows a terminal window titled "zhwang@zhwang-VirtualBox: ~". The window displays the contents of the `~/.bashrc` file. The code is as follows:

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
```

At the bottom right of the terminal window, there are two small buttons: "1,4" and "Top".

Figure 9: the `.bashrc` script

The figure shows two terminal windows. The top window displays a bash script named `bash_file.sh` with various annotations explaining its syntax:

```

1 #! /bin/bash
2 # Define a variable and echo it when executed from the terminal.
3 hello_world="This is a hello world from bash script!"
4 echo $hello_world ← Call a variable.
5 # Ask user for input, and append hello to the user input.
6 echo "What is your name?"
7 read userword ← Read the user input from cmd.
8 echo "$userword, a hello from bash script!" ← Call a command in the script
9 # check how many characters
10 letter_count=$(echo "$userword" | grep -o -i '[a-z]' | wc -l)
11 echo "you name has $letter_count characters!"
12 # response a conditional statement
13 if [ $letter_count -ge 5 ]; then → Writing a conditional statement.
14     echo "Your name is very long!"
15 else
16     echo "Your name is very short!" ← Terminate with fi
17 fi → Writing a loop (look at the arithmetic construct ((..))).
18 echo "Let me count again!" →
19 # count the words again with a loop
20 for ((i=1; i<=$letter_count; i++)); do →
21     echo $i → Terminate with done
22 done →

```

The bottom window shows the execution of the script:

```

zhang@zhang-VirtualBox:~/Desktop/bash_example$ chmod +x bash_file.sh
zhang@zhang-VirtualBox:~/Desktop/bash_example$ ./bash_file.sh
This is a hello world from bash script!
What is your name? zhang ← Ask for input.
zhang, a hello from bash script! → Changing permission to executable.
you name has 6 characters! → Execute the file.
Your name is very long!
Let me count again!
1
2
3
4
5
6

```

Figure 10: basics of bash scripts

```

1 #! /bin/bash
2
3 #user provided arguments
4 number1=$1           ← Read arguments
5 number2=$2           ← Check arguments
6 if ! [[ "$number1" =~ ^[0-9]+$ ]] || ! [[ "$number2" =~ ^[0-9]+$ ]]; then
7     echo "Error: Both arguments must be valid numbers."
8     exit 1 # Return an error code → If error, exit.
9 fi
10
11 echo "I got two numbers, $number1 and $number2"
12
13 # define a function that adds two numbers
14 add_func(){           → A function with two input
15     echo "adding..."   arguments.
16     local element1="$1" → Arguments defined locally
17     local element2="$2" within the function
18     local element_sum=$((element1+element2))
19     echo "result gives: $element_sum" → add within the arithmetic
20 }                      construct ((..)).
21
22 add_func number1 number2 → Function call in the script
23

```

Annotations on the right side of the code:

- Line 4: "Read arguments"
- Line 5: "Check arguments are numbers."
- Line 14: "A function with two input arguments."
- Line 16: "Arguments defined locally within the function"
- Line 18: "add within the arithmetic construct ((..))."
- Line 22: "Function call in the script"


```

zhang@zhang-VirtualBox:~/Desktop/bash_example$ chmod +x bash_function.sh
zhang@zhang-VirtualBox:~/Desktop/bash_example$ ./bash_function.sh 2 6
I got two numbers, 2 and 6
adding...
result gives: 8
zhang@zhang-VirtualBox:~/Desktop/bash_example$ ./bash_function.sh 2 zhang
Error: Both arguments must be valid numbers.
zhang@zhang-VirtualBox:~/Desktop/bash_example$ 

```

Annotations on the right side of the terminal output:

- Line 6: "Execute the file with input args."
- Line 18: "error here, exit and display the error message."

Figure 11: functions in bash scripts

The screenshot shows two terminal windows. The top window displays a bash script named `bash_filecheck.sh` with annotations explaining its logic:

```

2
3 dir=$1      ← Read arguments for all files in the dir.
4
5 for file in "$dir"/*; do
6     if [ -f "$file" ]; then → If it is a file (-f), echo it is
7         echo "A file $file"
8     else
9         echo "Not A file $file"
10    fi
11 done
12

```

The bottom window shows the execution of the script:

```

zhang@zhang-VirtualBox:~/Desktop/bash_example$ chmod +x bash_filecheck.sh
zhang@zhang-VirtualBox:~/Desktop/bash_example$ ./bash_filecheck.sh ~/Desktop
Not A file /home/zhang/Desktop/bash_example
A file /home/zhang/Desktop/Copy_P129_info.txt ← Check the desktop.
Not A file /home/zhang/Desktop/git_example
A file /home/zhang/Desktop/lecture_material
Not A file /home/zhang/Desktop/Mount_host
Not A file /home/zhang/Desktop/P129_Course
Not A file /home/zhang/Desktop/P129_data
A file /home/zhang/Desktop/P129_info.txt
Not A file /home/zhang/Desktop/P129_ps
A file /home/zhang/Desktop/wget-log
zhang@zhang-VirtualBox:~/Desktop/bash_example$ 

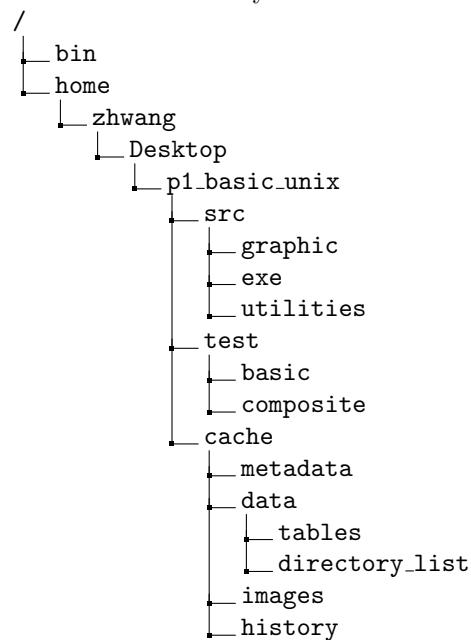
```

Figure 12: File management in bash

Problem 1: Basic UNIX File Management

In most projects, a directory tree is an essential tool for file management. Here, we will construct a directory tree and explore basic Linux commands. Below is an example of a complete directory tree, starting from the root “/”.

Please submit **all** your commands (including errors) in a text file named “P1_cmd_history.txt” in the directory “/p1_basic_unix”. For example, if you want to select the last 10 commands you entered and append them to a file, you can use the following ‘history’ command: “history | tail -n 10 >> P1_cmd_history.txt”. Please note that by doing this, the last command in the text file is **guaranteed** to be the ‘history’ command above.



A

1. Construct the directory tree in the container. Hint: you do not need to start from the root, instead, starting with host, so you can directly access files from your local computer!
2. Move to “/p1_basic_unix”, and add 3 **empty** “.txt” files: test_1.txt, test_2.txt, test_3.txt, to the directory ”/p1_basic_unix/test/basic”.
3. Stay in ”/p1_basic_unix”, and use the **echo** command to print your name and today’s date to test_1.txt.
4. Stay in ”/p1_basic_unix”, use the **cat** command to print test_1.txt to the terminal.

5. Stay in ”/p1_basic_unix”, use the **ls -l** command to print test_1.txt to test_2.txt.
6. Stay in ”/p1_basic_unix”, make the test_1.txt file executable with **chmod +x**, and **append** the result of **ls -l** to test_2.txt.

B

1. Download the compressed tar file ”P1_B.tar.gz” from the course website under Problem Sets, Downloads. Extract the tar file to the directory ”/p1_basic_unix”. Hint: use the ”wget ...” command. **Please check the sha256sum!**
2. Stay in ”/p1_basic_unix”, move example.sh from ”P1_B” to ”/p1_basic_unix”.
3. Stay in ”/p1_basic_unix”, rename two existing files in the directory ”P1_B” from example_1.txt and example_2.txt to demo_1.txt and demo_2.txt.
4. Stay in ”/p1_basic_unix”, move those two newly named files to ”/p1_basic_unix/src/exe” and ”/p1_basic_unix/cache/data/tables”.
5. Stay in ”/p1_basic_unix”, remove the directory ”P1_B”.
6. Stay in ”/p1_basic_unix”, create a symbolic link for the file demo_1.txt with the name demo_link.
7. Set an alias ”ll” that defines the operation ”ls -l”.

C

1. Stay in ”/p1_basic_unix”, use **grep** to search for the keyword ”statistics” in demo_1.txt, and using pipelines, print the result to a new file named ”grep_stat_demo1.txt”. Place the new file into the directory ”/p1_basic_unix/src/utilities”.
2. Stay in ”/p1_basic_unix”, perform a **grep** search for the keyword ’statistics’ across all directories. Utilize pipelines to redirect the results into a new file named ’grep_stat_all.txt,’ placing it within the directory ”/p1_basic_unix/cache/images”.
3. Stay in ”/p1_basic_unix”, use the **find** command to locate all files with a ”.txt” extension and print them to the test_3.txt file.
4. Stay in ”/p1_basic_unix”, employ both the **find** and **grep** commands to search for the keyword ’statistics’ within all files of the ’.txt’ format across all directories.

Submission to Github

For problem 1 submission, you are asked to create your own repository with git and push the “.tar.gz” file to the remote as an archive.

1. Using the **tar** command, compress the directory ”/p1_basic_unix” into a **single** ”.tar.gz” file named ”p1_basic_unix.tar.gz”.
2. Obtain the sha256sum by running ”sha256sum p1_basic_unix.tar.gz”.

Then, do the following for submitting your P1 to the remote.

1. Create a separate directory, /Archive_P1, and initialize git.
2. Copy your tar.gz file for the first problem to this repository.
3. Initialize the main branch, stage the change, and commit the change.
4. Following the discussion in the section, create your personal access token on github.
5. Commit to the remote repository. As what you did in the section, you need to first create the remote repository (with the same name) on github!

Problem 2: File Management and Bash Scripts

Bash scripts hold significant value in modern Linux system administration; they are used for automating tasks and performing various operations. In this problem, we will explore the fundamental uses of bash scripts.

In modern computing, file searching is critical for data extraction. The output data files from an instrument are typically labeled with information such as subject, year, month, date, and index (if there are multiple files). For example, in an electron scattering experiment, the instrument produces data files as binary files with names like:

```
example_file_name : electron_scattering_2023 - 10 - 04_sample_index_0.bin
```

In this question, you are will manage the data files created by your collaborator who measure the energy lost in an inelastic electron scattering experiment.

1. Fork the problem /P_2 on GitHub, and then clone the forked repository.
The directory tree has the following structure:

```
/  
└── P2  
    └── /Problem_1  
        └── /electron_scattering_data
```

2. Imagine your collaborator accidentally output data files to the wrong directory, i.e. /Problem_1. There are 500 files of them, and you need to delete them. **write a **single** line on the terminal that does the job.**
3. Your future collaborators may need to address this issue again, but they may have no knowledge of Linux commands. Write a bash script that defines an alias named 'file_remove.' It should accept a single argument: the name of the target directory. Hint: Use your previous work.
4. Navigate to the directory /electron_scattering_data. Your collaborator has requested that you separate the files with odd indexes from those with even indexes and move them to two new directories within the current directory. The two directories should be named /electron_scattering_data/odd and /electron_scattering_data/even, respectively. Write a bash script to accomplish this task. Hint: Use a for loop.
5. Submit a **pull request** on github. we will later review your code and give you comments on your submission.

(optional) If you are interested and have time, decode the binary files above with **python** and infer the corresponding statistical distributions.

Problem 3: Number Conversions and Bash Scripts

Although less common, people use bash scripts to perform basic calculations or conversions. There are three types of numbers: decimal (base-10), binary (base-2), and hexadecimal (base-16). An example conversion is provided in Table 1. In this problem, you are asked to **write a bash script that converts**

| Hexadecimal | Decimal | Binary |
|-------------|---------|----------|
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| A | 10 | 1010 |
| B | 11 | 1011 |
| C | 12 | 1100 |
| D | 13 | 1101 |
| E | 14 | 1110 |
| F | 15 | 1111 |
| 10 | 16 | 00010000 |
| 1A | 26 | 00011010 |
| 1F | 31 | 00011111 |
| 20 | 32 | 00100000 |

Table 1: Examples on Hexadecimal, Decimal, and Binary Number Conversions

| Division | Quotient | Remainder |
|-------------|----------|-----------|
| 125064 / 16 | 7816 | 8 |
| 7816 / 16 | 488 | 8 |
| 488 / 16 | 30 | 8 |
| 30 / 16 | 1 | 14 |
| 1 / 16 | 0 | 1 |

Table 2: Example: converting Decimal to hexadecimal via division remainders. In hexadecimal, it is 1E888. Or in binary, it is 11110100010001000.

a given decimal number to both hexadecimal and binary numbers. In addition, the program should only accept decimal values smaller than 100000. You are required to hard-code the conversions using information from both Table. 1 and Table. 2! Please copy your bash script into a directory named /Problem_3, and the outcome should be passed

to a .txt file with name “conversion_result.txt”, also in /Problem_3.

Push it to github with the repo name /Problem_3.

Problem 4: Version Control with Git.

Similar to what we have done in the section and write the answer in a README file under the repository you created in the section.

1. Checkout a different branch, called feature_branch. Create a readme file with “hello world!”. Then, stage, commit and push to the **feature_branch** and push to the remote.
2. After you push the feature_branch, merge the feature_branch with the main branch locally, and push to the remote. What do you see locally and remotely on github?
3. Modify the text in readme file from ”hello world!” to ”hello world from remote!” **on github manually**, and pull from the remote origin. Do you see the update?