



CyberCamp at UNK

Bash: general commands and scripts

What you will learn with this tutorial...

*This tutorial focus on teaching to you an important tool present on Linux systems: the **Bash**. First of all, you will understand what bash is and what are its most common used commands. After that you will learn what is a **bash script**, another useful resource available on Linux systems. As always, at the end, you will have an exercise to practice what you just learnt. Let's start!*

1 What is bash?

Bash is a *command language* written for Unix systems. Command languages are languages that can be used directly at a command line, the users typically interact with these languages using a *terminal* emulator. All Linux based systems have a bash terminal that can be accessed by the users. Terminal commands have a lot of advantages, like:

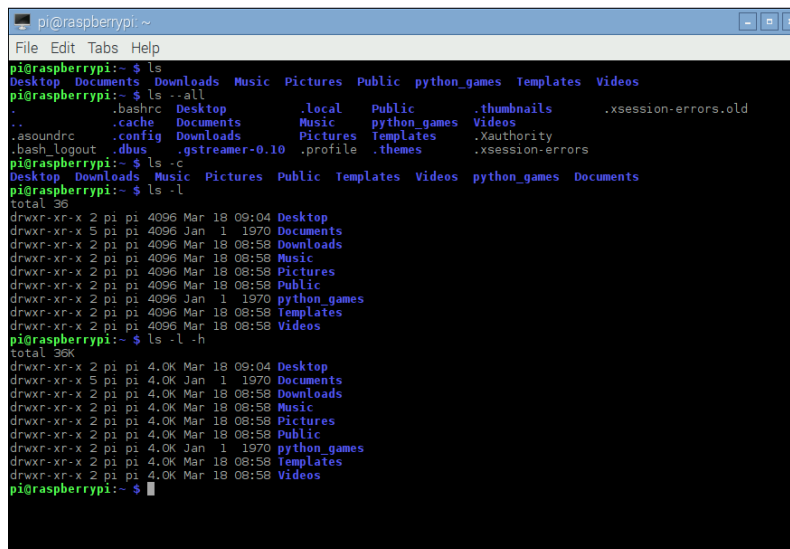
- Bash commands and scripts (we will discuss soon what a *bash script* is) that run in one machine will run in all bash based machines;
- Practically everything on the system can be configured throughout command line;
- They allow the user to automate long and repetitive tasks.

These are some of the reasons why know some bash commands are really useful and important. What we are going to do now is to learn some of the most used bash commands, their names, definitions, and some practical examples.

2 Bash commands

In this section you will be presented to the most common and useful bash commands. Bash has **a lot** of commands and we simply couldn't talk about all of them here. What we will focus on will be the most basic ones (but still really important). We will see how to navigate through our files and directories, list, copy, move and delete our files, and also how to perform searches, all of this just using typed commands in our terminal. Below you will have a series of items, each item will talk about a certain command, what it does and simple examples of how to use it. So open your terminal, read those commands and try them out for yourself!

- **ls**: List all the files of a directory. If you don't specify a path, list all the files of the current directory.
 - **ls *directorypath***: list all the files of the given directory path;
 - **ls --all**: list all the files, including the hidden ones (hidden files start with a dot);
 - **ls -c**: list the sorted files by time of modification;
 - **ls -l**: list the files in a list format;
 - **ls -l -h**: list the files in a list format and their size in a human readable format.



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~$ ls  
Desktop Documents Downloads Music Pictures Public python_games Templates Videos  
pi@raspberrypi:~$ ls --all  
.. .bashrc Desktop Local Public .thumbnails .xsession-errors.old  
.. .cache Documents Music python_games Videos  
.. .config Downloads Pictures Templates .Xauthority  
.. .bash_logout .dbus .gstreamer-0.10 .profile .themes .xsession-errors  
pi@raspberrypi:~$ ls -c  
Desktop Downloads Music Pictures Public Templates Videos python_games Documents  
pi@raspberrypi:~$ ls -l  
total 36  
drwxr-xr-x 2 pi pi 4096 Mar 18 09:04 Desktop  
drwxr-xr-x 5 pi pi 4096 Jan 1 1970 Documents  
drwxr-xr-x 2 pi pi 4096 Mar 18 08:58 Downloads  
drwxr-xr-x 2 pi pi 4096 Mar 18 08:58 Music  
drwxr-xr-x 2 pi pi 4096 Mar 18 08:58 Pictures  
drwxr-xr-x 2 pi pi 4096 Mar 18 08:58 Public  
drwxr-xr-x 2 pi pi 4096 Jan 1 1970 python_games  
drwxr-xr-x 2 pi pi 4096 Mar 18 08:58 Templates  
drwxr-xr-x 2 pi pi 4096 Mar 18 08:58 Videos  
pi@raspberrypi:~$ ls -l -h  
total 36K  
drwxr-xr-x 2 pi pi 4.0K Mar 18 09:04 Desktop  
drwxr-xr-x 5 pi pi 4.0K Jan 1 1970 Documents  
drwxr-xr-x 2 pi pi 4.0K Mar 18 08:58 Downloads  
drwxr-xr-x 2 pi pi 4.0K Mar 18 08:58 Music  
drwxr-xr-x 2 pi pi 4.0K Mar 18 08:58 Pictures  
drwxr-xr-x 2 pi pi 4.0K Mar 18 08:58 Public  
drwxr-xr-x 2 pi pi 4.0K Jan 1 1970 python_games  
drwxr-xr-x 2 pi pi 4.0K Mar 18 08:58 Templates  
drwxr-xr-x 2 pi pi 4.0K Mar 18 08:58 Videos  
pi@raspberrypi:~$
```

Figure 1: ls command examples

- **cd:** This is a important command used to navigate through directories.
 - **cd *directoryname*:** to enter into a subdirectory;
 - **cd *..*:** to navigate up one directory level;
 - **cd *-*:** to navigate to the previous directory;
 - **cd */*:** to navigate to the root directory;
 - **cd:** to navigate to the home directory.
- **mkdir:** Create a new directory.
 - **mkdir *direcotryname*:** create a directory with the name given by the user. Note in the image below that when we type: `mkdir Bash Tutorial` two different directories are created, `Bash` and `Tutorial`. If we want to create a directory that is composed by more than one word we must use double quotes, hence `mkdir "Bash Tutorial"`. Same thing applies when using `cd`. The access for this directory would be `cd "Bash Tutorial"`.

```

pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~$ ls
Desktop Documents Downloads Music Pictures Public python_games Templates Videos
pi@raspberrypi:~$ cd Documents
pi@raspberrypi:~/Documents$ ls
BlueJ Projects Greenfoot Projects Scratch Projects
pi@raspberrypi:~/Documents$ cd ..
pi@raspberrypi:~$ ls
Desktop Documents Downloads Music Pictures Public python_games Templates Videos
pi@raspberrypi:~$ mkdir Bash Tutorial
pi@raspberrypi:~$ ls
Bash Desktop Documents Downloads Music Pictures Public python_games Templates Tutorial Videos
pi@raspberrypi:~$ mkdir "Bash Tutorial"
pi@raspberrypi:~$ ls
Bash Bash Tutorial Desktop Downloads Pictures python_games Tutorial
Bash Tutorial Documents Music Public Templates Videos
pi@raspberrypi:~$

```

Figure 2: `cd` and `mkdir` command examples

- **pwd:** Command used to know in which directory you are in.
 - **pwd:** show the full path of the current directory.
- **cp:** Command used to copy files and diretories.
 - **cp *file.txt* *copyfile.txt*:** Copy files. Create a copy of `file.txt` named `copyfile.txt` in the same directory;
 - **cp *-r* *directory* *directorycopy*:** Copy directories. Create a copy of `directory` named `directorycopy` in the same directory.
- **mv:** Command used to rename or move files.
 - **mv *oldname.txt* *newname.txt*:** Rename a file;
 - **mv *file.txt* *~/Desktop*:** Move a file from the current directory to a different location (in this case the `Desktop` directory).
- **rm:** Remove or delete files and directories.

- **rm filename.txt**: Delete a file;
- **rm -r directoryname**: Delete a directory.
- **sudo**: Used to perform root user commands as installing new programs or performing files operations outside of your user home directory.
 - **sudo su**: Allow the user to login as root user of the system.
- **cat**: This command is used to display text files on screen, copy text files, combine text files, create new text files.
 - **cat /etc/passwd**: Print the contents of a file on the directly on the terminal;
 - **cat /etc/passwd > test.txt**: Create a new file with the output of the cat command;
 - **cat file1.txt file2.txt > concatenated.txt**: Concatenate file1.txt content with file2.txt content into a new file called concatenated.txt;
 - **cat oldfile.txt > newfile.txt**: Copy a file, just like the command cp;
 - **cat -n /etc/passwd**: Number the output lines of the cat command.

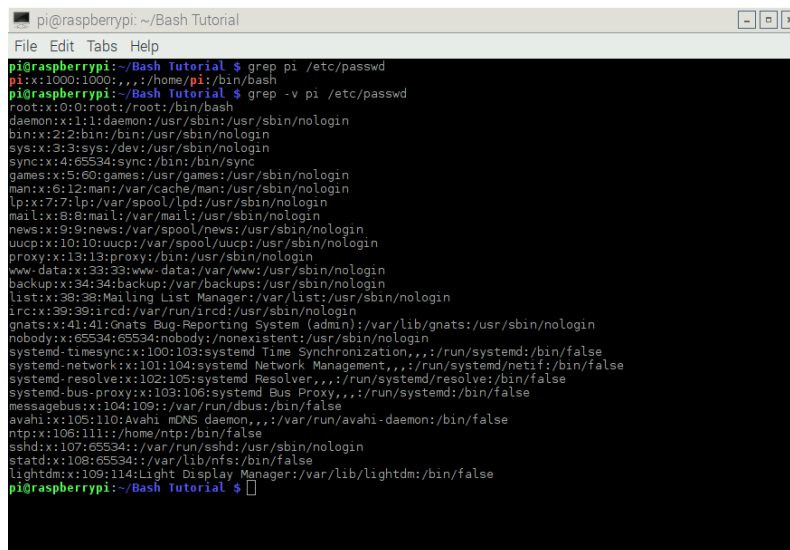
```

pi@raspberrypi:~/Bash Tutorial
File Edit Tabs Help
pi@raspberrypi:~/Bash Tutorial $ cat > myfile.txt
Hello Bash!
^C
pi@raspberrypi:~/Bash Tutorial $ ls
myfile.txt
pi@raspberrypi:~/Bash Tutorial $ cat myfile.txt
Hello Bash!
pi@raspberrypi:~/Bash Tutorial $ cat myfile.txt > newfile.txt
pi@raspberrypi:~/Bash Tutorial $ ls
myfile.txt  newfile.txt
pi@raspberrypi:~/Bash Tutorial $ cat newfile.txt
Hello Bash!
pi@raspberrypi:~/Bash Tutorial $ cat myfile.txt newfile.txt > concatenated.txt
pi@raspberrypi:~/Bash Tutorial $ ls
concatenated.txt  myfile.txt  newfile.txt
pi@raspberrypi:~/Bash Tutorial $ cat concatenated.txt
Hello Bash!
Hello Bash!
pi@raspberrypi:~/Bash Tutorial $

```

Figure 3: cat command examples

- **grep**: Command used for searches. It can find text or look for lines containing matches of words and strings in the given file. It can also be used together with regular expressions.
 - **grep pi /etc/passwd**: Search for the user pi at the system users file. Returns to the output just the matching lines;
 - **grep -i pi /etc/passwd**: Does the same as the above, but -i forces grep to ignore case sensitive, so now grep will search for PI, pi, Pi, pI;
 - **grep -i -v pi /etc/passwd**: -v inverts the match, so now the command will return all the lines that does not contain pi and its variations.

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Bash Tutorial'. The terminal shows the execution of two commands. The first command is 'grep pi /etc/passwd', which outputs the password entry for the user 'pi'. The second command is 'grep -v pi /etc/passwd', which outputs all other entries in the /etc/passwd file, excluding the 'pi' user. The output of the second command is a long list of system and user accounts, including root, daemon, bin, sys, sync, games, man, lp, mail, news, uucp, proxy, www, backup, list, irc, gnats, nobody, systemd-timedate, systemd-networkd, systemd-resolved, systemd-bus-proxy, messagebus, avahi, ntp, sshd, statd, and lightdm.

```
pi@raspberrypi:~/Bash Tutorial
File Edit Tabs Help
pi@raspberrypi:~/Bash Tutorial $ grep pi /etc/passwd
pi:x:1000:1000:,,,:/home/pi:/bin/bash
pi@raspberrypi:~/Bash Tutorial $ grep -v pi /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mail List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time Synchronization,,,:/run/systemd:/bin/false
systemd-networkd:x:101:104:systemd Network Management,,,:/run/systemd/netif:/bin/false
systemd-resolved:x:102:105:systemd Resolver,,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:106:systemd Bus Proxy,,,:/run/systemd:/bin/false
messagebus:x:104:109:/:/var/run/dbus:/bin/false
avahi:x:105:110:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
ntp:x:106:111:/:/home/ntp:/bin/false
sshd:x:107:65534:/:/var/run/sshd:/usr/sbin/nologin
statd:x:108:65534:/:/var/lib/nfs:/bin/false
lightdm:x:109:114:Light Display Manager:/var/lib/lightdm:/bin/false
pi@raspberrypi:~/Bash Tutorial $
```

Figure 4: grep command examples

As discussed in the beginning of this section, these are just a few of a lot of existing bash commands. One really useful command that was not shown before is the **man** command. This is a user manual that is by default built into most Linux. If you are not sure what a command does, or how to use it, all you have to do is type:

```
man commandname
```

This is a really useful command that you should memorize, it will for sure help you a lot specially if you are starting using bash now.

Now that you know the basics of bash commands, let's understand what bash scripts are.

3 Bash scripts

You just learned some bash commands that run directly in your terminal. A **bash script** is a plain text file with a series of these commands. Every command you can run on the command line can be put into a script and will do exactly the same thing. Then why create scripts when you can run all the commands in the terminal?

You can use scripts to automate long and repetitive tasks, this way you don't have to type all the commands again every time you need to perform same tasks you have already done before.

In the subsection below you will code your first bash script and learn how to execute it.

3.1 How to execute scripts

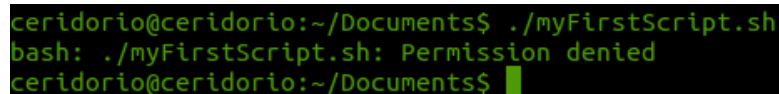
By convention, scripts have a `.sh` extension. Let's write a simple bash script, then understand what it does and learn how to run it. Open a plain text file, type the code below and save it on your *Documents* folder as `myFirstScript.sh`.

```
1  #!/bin/bash
2
3  # first Bash script
4  echo Hello World!
```

To execute your script navigate to your *Documents* directory and type the following command:

```
./myFirstScript.sh
```

You will probably receive the following error message:



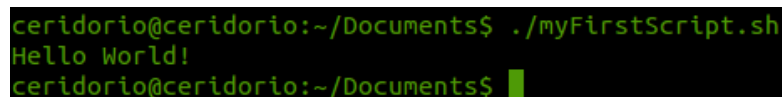
```
ceridorio@ceridorio:~/Documents$ ./myFirstScript.sh
bash: ./myFirstScript.sh: Permission denied
ceridorio@ceridorio:~/Documents$
```

Figure 5: permission denied when running a script

This happens because before executing scripts we need to grant *permission* to the file. In your *Documents* directory type:

```
chmod 775 myFirstScript.sh
```

This command will allow the owner to write or modify the file and all the users will be able to execute it. Now run your execution command again, you should obtain:



```
ceridorio@ceridorio:~/Documents$ ./myFirstScript.sh
Hello World!
ceridorio@ceridorio:~/Documents$
```

Figure 6: your first bash script in execution

You just completed your first bash script! Let's understand our code:

- **Line 1:** This tells our script the path to the interpreter that should be used to run the rest of the lines. In this case we are indicating to the script the path to the bash location in our system.
- **Line 3:** Just like Python, comments here start with `#`.
- **Line 4:** The `echo` command prints on the terminal screen the string after it, in this case `Hello World!`

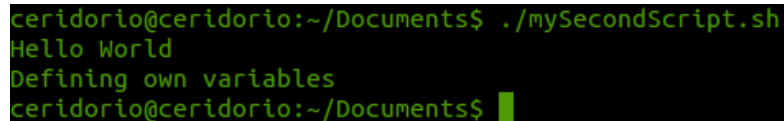
Another important thing we can do with scripts is to create and manipulate variables. That's what we are going to see next.

3.2 Defining own variables

We will now take a quick look at how to define our own variables inside scripts. Create a new file on your *Documents* directory called `mySecondScript.sh` and type:

```
1  #!/bin/bash
2
3  # defining our own variables
4  firstVariable=Hello
5  secondVariable=World
6  thirdVariable="Defining own variables"
7
8  echo $firstVariable $secondVariable
9  echo $thirdVariable
```

Once again, grant permission to your file and execute it:



```
ceridorio@ceridorio:~/Documents$ ./mySecondScript.sh
Hello World
Defining own variables
ceridorio@ceridorio:~/Documents$ █
```

Figure 7: setting own variables example

Let's understand some important lines from our code:

- **Line 4, 5:** We define two variables, `firstVariable` and `secondVariable` and we assign them one word each. Note that **there is no space** before and after the equal sign, if you put a space here your script will not execute (try it yourself!).
- **Line 6:** We define another variable, `thirdVariable`, but this time we want a sentence as its value. To do so we must put our sentence inside double quotes, otherwise, once again, our script will throw errors.
- **Line 8, 9:** We simply display the values of the variables. To access the value of a variable you type `$` followed by the name of the variable (here again, without any space between them).

This is how we can define our own variables with bash script. However, bash script has some special variables ready for our use. This is what we will study now.

3.3 Some special variables

Bash script has quite a variety of variables already defined that we can use in our scripts. These are some of them:

- **\$0:** The name of the bash script;
- **\$1 - \$9:** The first 9 arguments passed when executing the bash script;
- **\$USER:** The username of the user running the script;
- **\$HOSTNAME:** The hostname of the machine running the script;
- **\$RANDOM:** Always return a different number when referred to.

As stated above, these are just some of the variables available, if you wish to know more of them a quick web search can give you the complete list!

We are going to build now another example, this time using the already defined script variables. This example has no practical value, we could execute it directly on the command line and that would be easier and faster. However, the purpose now is to understand how we can use the special variables in our scripts. Create a new file on your *Documents* directory called `myThirdScript.sh` and type:

```
1  #!/bin/bash
2
3  #using some special variables
4  echo
5  echo "User executing the script: $USER"
6  echo
7  echo "Files before executing command:"
8
9  ls
10 mkdir $1 $2
11
12 echo
13 echo "Files after executing command:"
14
15 ls
16
17 echo
```

This should be your output:


```
ceridorio@ceridorio:~/Documents$ ./myThridScript.sh firstDir secondDir
User executing the script: ceridorio
Files before executing command:
myFirstScript.sh mySecondScript.sh myThridScript.sh
Files after executing command:
firstDir myFirstScript.sh mySecondScript.sh myThridScript.sh secondDir
ceridorio@ceridorio:~/Documents$ █
```

Figure 8: using defined variables

The code is really straightforward. Look at the execution. Now when we execute our script we call its file with `./myThridScript.sh` and in the front we inform two more arguments, that will be used in our script. What our script does is to list the files in the directory before and after creating two more directories. These directories will receive the names passed as arguments in the moment we called the script. Note on **line 5** that we also use another special variable, `$USER` inside another `echo` string. Whenever you want to concatenate text with variable values you can use this technique.

This is the basics of bash scripting. There are a lot more resources you can use that we will not discuss here, as for example user input, conditionals and loops (just like in Python). This is a basic guide so you can familiarize yourself with the scripting language. If you really enjoyed it be sure to search a little more about it on the Internet! Bash scripting is really powerful and useful inside Unix environments.

Now that you already learned the most useful commands and how to apply them inside scripts, move to the exercise section!

Now try for yourself!

Your job is to create a simple bash script using the list of commands that you now have and your knowledge about some special bash variables.

Your script should be really easy, any complicated command will be needed. It must take two arguments, the first one will be a word of your choice that will be the name of one directory. The second argument will be a username that you will use to apply some searches at the usernames file of your system. Here are some steps defining what your code should do:

- 1. Show on the screen the username and the hostname of the machine executing the script;*
- 2. Create a directory with the name given in the first argument and navigate into it;*
- 3. Search in the file **/etc/passwd** for the user passed as the second argument and store the result of the search in a new file called **searchOne.txt**;*
- 4. Show the content of **searchOne.txt**;*
- 5. Do a reverse search in the file **/etc/passwd** also with the second argument and store the result of the search in a new file called **searchTwo.txt**;*
- 6. Concatenate **searchOne.txt** and **searchTwo.txt** in a new file called **concatenated.txt**;*
- 7. Show the content of **concatenated.txt**.*

In the next page you can check an example of how your script should look like when executed.

```

pi@raspberrypi: ~/Documents
File Edit Tabs Help

pi@raspberrypi:~/Documents$ ./bashExercise.sh myScript pi
User executing the script: pi
Host executing the script: raspberrypi

Creating directory with name passed as first argument...
Searching for user passed as second argument.
Result saved in searchOne.txt

Content of searchOne.txt:
pi:x:1000:1000:::/home/pi:/bin/bash

Reverting the search passed as second argument.
Result saved in searchTwo.txt

Concatenating the searches.

Concatenated file:
pi:x:1000:1000:::/home/pi:/bin/bash
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:103:systemd Time Synchronization,,:/run/systemd:/bin/false
systemd-networkd:x:101:104:systemd Network Management,,:/run/systemd/netif:/bin/false
systemd-resolve:x:102:105:systemd Resolver,,:/run/systemd/resolve:/bin/false
systemd-bus-proxy:x:103:106:systemd Bus Proxy,,:/run/systemd:/bin/false
messagebus:x:104:109:/var/run/dbus:/bin/false
avahi:x:105:110:Avahi mDNS daemon,,:/var/run/avahi-daemon:/bin/false
ntp:x:106:111:/home/ntp:/bin/false
sshd:x:107:65534:/var/run/sshd:/usr/sbin/nologin
statd:x:108:65534:/var/lib/nfs:/bin/false
lightdm:x:109:114:Light Display Manager:/var/lib/lightdm:/bin/false
pi@raspberrypi:~/Documents$

```

Figure 9: exercise output sample

After the execution of your script the generated directory should contain the following files:

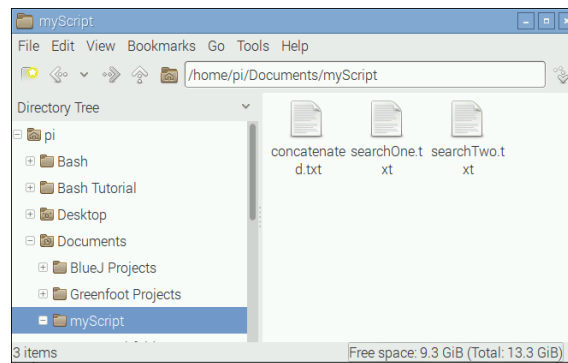


Figure 10: directory after script execution

Good luck!