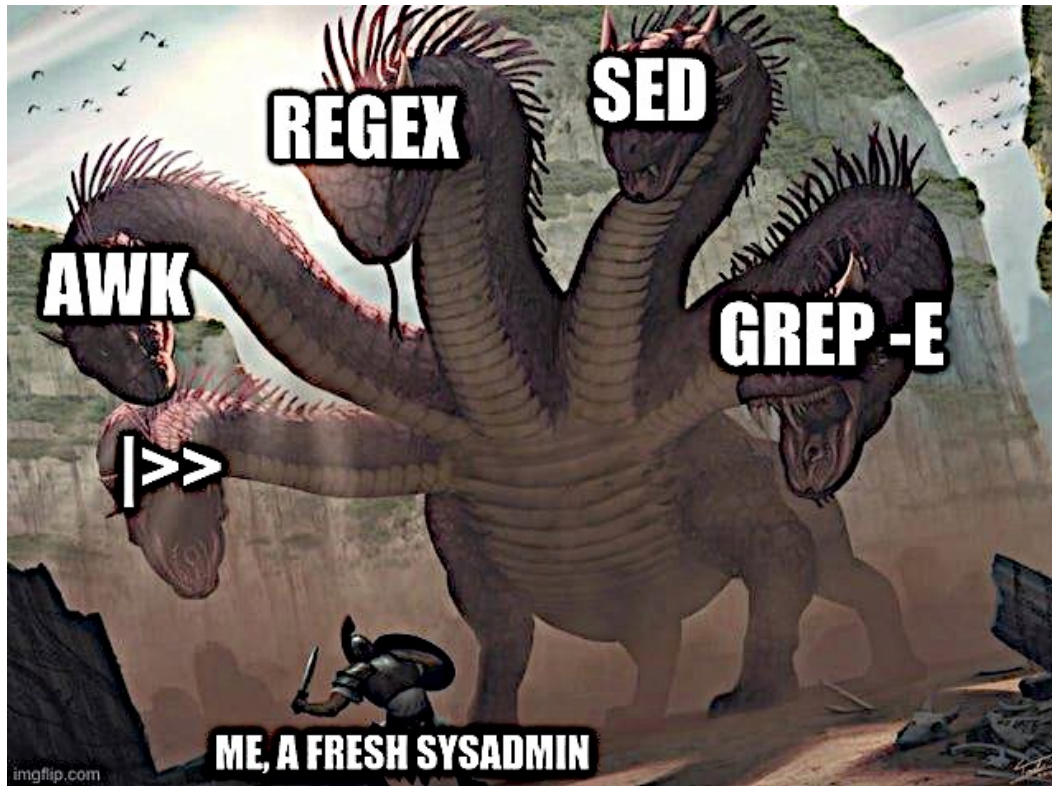


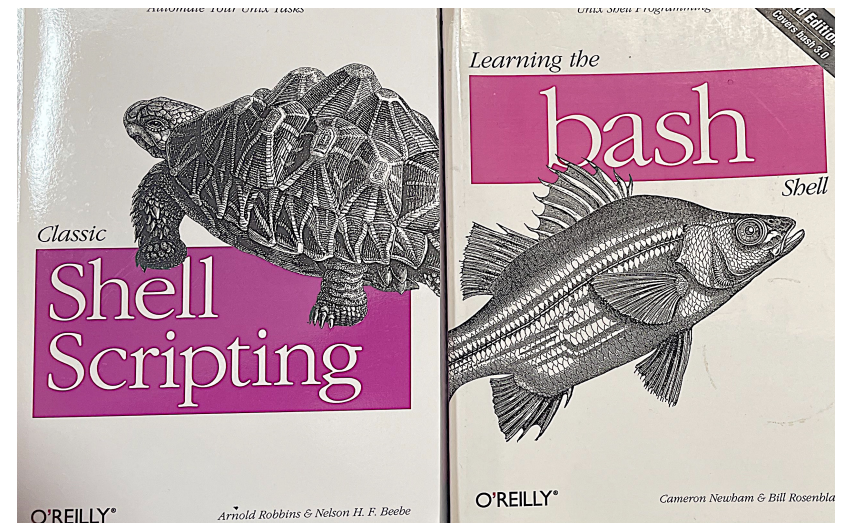
Lecture 3: Bash Language Part III

Photo of a new
Linux user learning
the command line



Suggested Literature

Both books are available at
<https://learning.oreilly.com>



RECAP: Creating New Directories

New directories can be created with the **mkdir** (make directory)

Syntax:

mkdir *<name_of_new_directory>*

Example:

mkdir codes

Creates a new directory named **codes**

```
whovian@phy504:~$ mkdir --help
```

```
Usage: mkdir [OPTION]... DIRECTORY...
```

```
Create the DIRECTORY(ies), if they do not already exist.
```

```
Mandatory arguments to long options are mandatory for short options
```

```
-m, --mode=MODE    set file mode (as in chmod), not a=rwx - umask
```

```
-p, --parents       no error if existing, make parent directories as
```

```
-v, --verbose       print a message for each created directory
```

```
-Z                  set SELinux security context of each created  
to the default type
```

```
--context[=CTX]    like -Z, or if CTX is specified then set the  
or SMACK security context to CTX
```

```
--help             display this help and exit
```

RECAP: The **cp** and **mv** commands

cp or **mv** commands accept new filenames or directories (*dir*) as their second args. If *dir*, then type relative or absolute paths

```
whovian@phy504:~/host$ ls
es.txt
whovian@phy504:~/host$ cp es.txt es2.txt
whovian@phy504:~/host$ mkdir test
whovian@phy504:~/host$ cp es.txt ./test/
whovian@phy504:~/host$ ls
es2.txt es.txt test
whovian@phy504:~/host$ ls ./test/
es.txt
whovian@phy504:~/host$ ls
es2.txt es.txt test
whovian@phy504:~/host$ mkdir test2
whovian@phy504:~/host$ mv es2.txt test2/
whovian@phy504:~/host$ ls
es.txt test test2
whovian@phy504:~/host$ ls ./test2/
es2.txt
whovian@phy504:~/host$
```

cp or **mv** commands have many options. Try

cp --help

and

mv --help

Copy directories

cp -r

one dash '-' and **r**

HW: research one vs two dashes on options

RECAP: Absolute Pathnames

- **Absolute paths** can be used to change working directories. Try this:

cd /

This command changes your working directory to the root dir

Verify this by executing the **pwd** command!

Return to your home directory? **cd** command with no arguments

```
whovian@phy504:~$ pwd
/home/whovian
whovian@phy504:~$ cd /
whovian@phy504:/$ pwd
/
```

pwd provides the absolute path of
the current directory

```
whovian@phy504:/$ ls
bin  dev  home  lib32  libx32  mnt  opt  root  sbin  sys  usr
boot  etc  lib  lib64  media  NGC-DL-CONTAINER-LICENSE  proc  run  srv  tmp  var
whovian@phy504:/$ cd
whovian@phy504:~$ pwd
/home/whovian
```

ls has many interesting options

```
whovian@phy504:~$ ls --help
```

```
Usage: ls [OPTION]... [FILE]...
```

```
List information about the FILES (the current directory by default).
```

```
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.
```

check **ls --help**

RECAP: Relative Pathnames

Sometimes, absolute paths are the quickest way to refer to a file/dir

The absolute path to a file or directory may be long!

```
RiverSong:Vivian vivianmiranda$ ls
Docker          PHY504.pptx
RiverSong:Vivian vivianmiranda$ pwd
/Users/vivianmiranda/Library/CloudStorage/OneDrive-StonyBrookUniversity/teaching/Phy504/Vivian
```

My laptop

This could make commands unwieldy! Is there an easier way?

A **relative path** or **relative pathname** specifies the position of a file/dir relative to some other entity the shell knows about

```
RiverSong:Vivian vivianmiranda$ ls ../Docker/
RiverSong:Vivian vivianmiranda$ ls ~/
Applications
Applications (Parallels)
```

from current dir OneDrive - Stony Brook University

```
RiverSong:Vivian vivianmiranda$ ls ../
PHY277          PHYS305_2022    Shared
RiverSong:Vivian vivianmiranda$ ls ../../
Chicago DL      Phy504  SM.pdf  books
RiverSong:Vivian vivianmiranda$
```

from parent directory Vivian bash

from grandparent directory

RECAP: The **ls** command

ls displays all files in the current directory (with information)

- **-d** list directories, not files (**options can tweak command behavior**)
- **-S** (yes CAPS): sort files by size, largest first
- **-t** sort by modification time, newest first
- **-U** (yes CAPS) do not sort
- **-l** use long listing format (with size in bytes)
- **-h** (human-readable)! That means converting bytes to Kb, Mb, Gb

```
whovian@phy504:~/host$ ls -Shl
```

```
total 428K
```

```
-rw-r--r-- 1 whovian users 175K Jan 13 00:22 hamlet.txt
-rw-r--r-- 1 whovian users 124K Jan 12 02:35 es.txt
-rw-r--r-- 1 whovian users 556 Jan 12 21:21 compile_des_y3
-rw-r--r-- 1 whovian users 416 Jan 13 00:38 wf.sh
-rw-r--r-- 1 whovian users 391 Jan 17 16:45 employees.txt
-rw-r--r-- 1 whovian users 342 Jan 20 01:22 address_list.txt
-rw-r--r-- 1 whovian users 341 Jan 20 01:06 address_list_backup.txt
-rw-r--r-- 1 whovian users 326 Jan 17 04:02 caseloop.sh
-rw-r--r-- 1 whovian users 283 Jan 17 03:31 whileloop3.sh
```

Yes! Concatenate options

RECAP: Deleting Files/Dir

Directories can be removed (deleted) with the **rm** (Remove file)

BE EXTREMELY CAREFUL WITH **rm** COMMAND

Syntax:

rm *<name_of_new_file>*

Example:

rm codes.txt

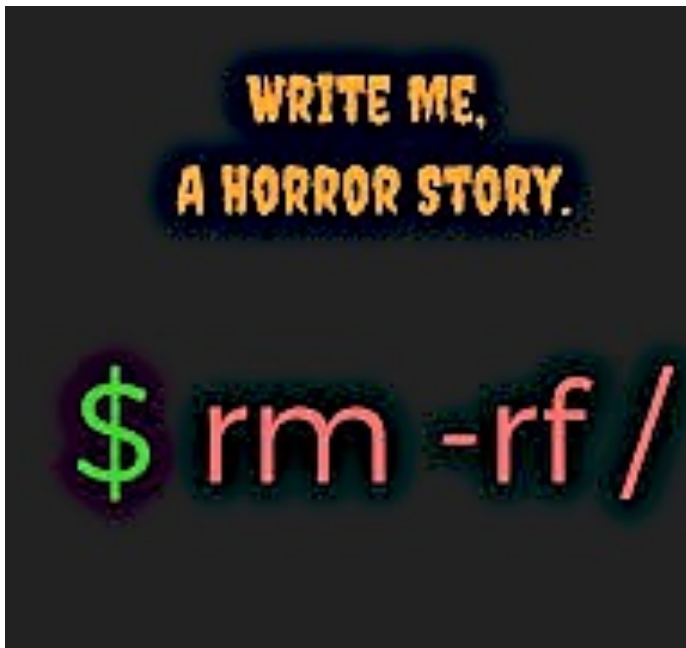
Deletes the file named **codes**

BE CAREFUL WITH **rm**

This command deletes

ALL your **files/dir**

-r (dir), **-f** (force)



RECAP: The **head** command

- The **-n** option also accepts negative integers!

```
whovian@phy504:~/host$ more singers.txt
4 Avril_Lavigne.
10 Rihanna.
2 Lady_Gaga.
5 Usher
14 Shakira
1 Justin_Timberlake.
20 Billie_Joe_Armstrong
3 Taylor_Swift.
whovian@phy504:~/host$ head -n -2 singers.txt
4 Avril_Lavigne.
10 Rihanna.
2 Lady_Gaga.
5 Usher
14 Shakira
1 Justin_Timberlake.
```

Negative number (example -2) = print ALL but the last two

RECAP: The **tail** command

Finding in real time who is trying to **hack** my server

```
vivianmiranda@amypond:~$  
vivianmiranda@amypond:~$ tail -f -n 0 /var/log/auth.log  
Jan 26 14:32:50 amypond sshd[360949]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=202.51.74.123 user=root  
Jan 26 14:32:52 amypond sshd[360949]: Failed password for root from 202.51.74.123 port 50690 ssh2  
Jan 26 14:32:53 amypond sshd[360949]: Received disconnect from 202.51.74.123 port 50690:11: Bye Bye [preauth]  
Jan 26 14:32:53 amypond sshd[360949]: Disconnected from authenticating user root 202.51.74.123 port 50690 [preauth]  
Jan 26 14:32:55 amypond sshd[360977]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=65.21.2.173 user=root  
Jan 26 14:32:57 amypond sshd[360977]: Failed password for root from 65.21.2.173 port 59946 ssh2  
Jan 26 14:32:58 amypond sshd[360977]: Received disconnect from 65.21.2.173 port 59946:11: Bye Bye [preauth]  
Jan 26 14:32:58 amypond sshd[360977]: Disconnected from authenticating user root 65.21.2.173 port 59946 [preauth]  
Jan 26 14:33:12 amypond sshd[361035]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=43.135.163.98 user=root  
Jan 26 14:33:12 amypond sshd[361033]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh  
ruser= rhost=81.182.248.193 user=root  
Jan 26 14:33:15 amypond sshd[361035]: Failed password for root from 43.135.163.98 port 56836 ssh2  
Jan 26 14:33:15 amypond sshd[361033]: Failed password for root from 81.182.248.193 port 47134 ssh2  
Jan 26 14:33:15 amypond sshd[361035]: Received disconnect from 43.135.163.98 port 56836:11: Bye Bye [preauth]  
Jan 26 14:33:15 amypond sshd[361035]: Disconnected from authenticating user root 43.135.163.98 port 56836 [preauth]  
Jan 26 14:33:15 amypond sshd[361033]: Received disconnect from 81.182.248.193 port 47134:11: Bye Bye [preauth]  
Jan 26 14:33:15 amypond sshd[361033]: Disconnected from authenticating user root 81.182.248.193 port 47134 [preauth]
```

That is one reason why you can't have a weak password. If your laptop connects to the internet, it is vulnerable to hacking.

Talking about hacking: **ssh** and keys

We learned in Lecture 1 that we can **SSH** to a server via

user1@machine1:frompath

Your credential is authenticated via a password that is sent to the server via an encrypted channel (**SSH** protocol ensures that)



This system still has weak protection! Why?

Because someone (or a hidden camera) can see you typing the password!

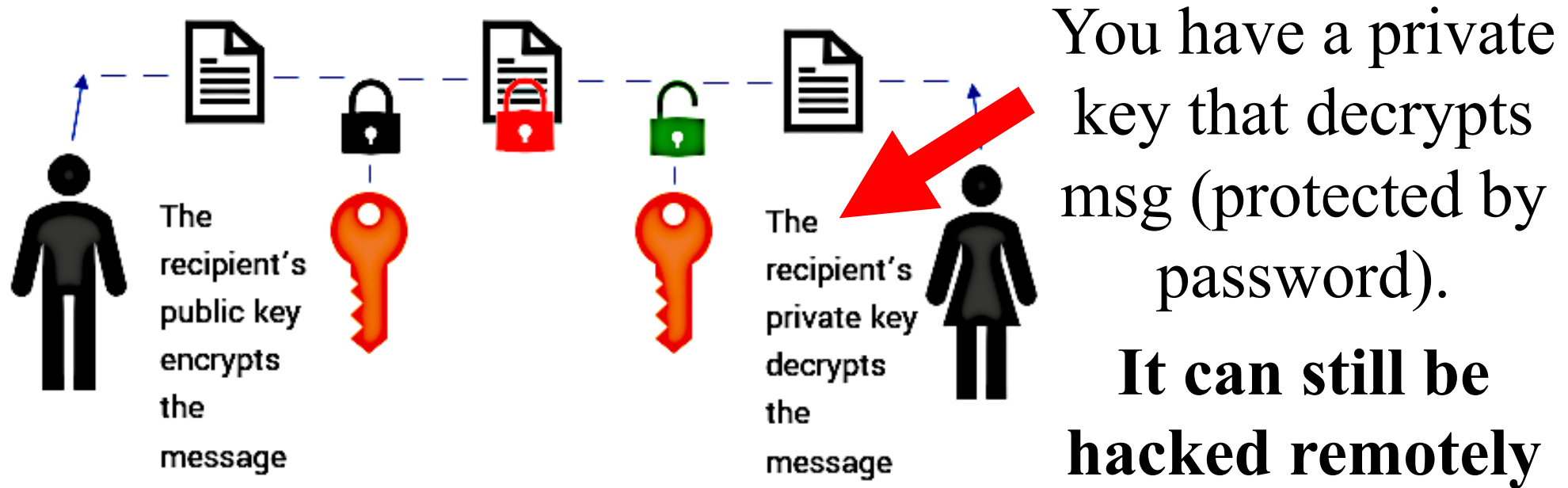
You need to have something hard to acquire

Talking about hacking: **ssh** and keys

We learned in Lect. 1 that we can **SSH** to a server via

user1@machine1:frompath

...something hard to acquire. Example: public-private key



Send everyone a public key that can encrypt a message in a way that can only be decrypted by your private key.

How? Via the **ssh-keygen** command ([hyperlink here](#))

Talking about hacking: **ssh** and smartcards

We learned in Lect. 1 that we can **SSH** to a server via

user1@machine1:frompath

On a high-security system – you may also need a smartcard



You have a password, a public-private key, and a smart card.

So, the hacker needs to check your password and steal your private key and **physical** card.

The **join** command

For pair of input lines with identical fields, write a line to output

```
whovian@phy504:~/host$ more foodtypes.txt
1 Protein
2 Carbohydrate
3 Fat
whovian@phy504:~/host$ more foods.txt
1 Cheese
2 Potato
3 Butter
whovian@phy504:~/host$ join foodtypes.txt foods.txt
1 Protein Cheese
2 Carbohydrate Potato
3 Fat Butter
```

```
whovian@phy504:~/host$ more foods2.txt
2 Potato
1 Cheese
3 Butter
whovian@phy504:~/host$ join foodtypes.txt foods2.txt
join: foods2.txt:2: is not sorted: 1 Cheese
2 Carbohydrate Potato
3 Fat Butter
```

Attention: field
\$1 needs to be
sorted!

The **join** command x2

For a pair of input lines with identical fields, write a line to output. What if the identical field is not column 1?

In that case, use the command options

-1 COL_FILE_1 -2 COL_FILE_2

```
whovian@phy504:~/host$ more wine.txt
```

```
White Reisling Germany
```

```
Red Rioja Spain
```

```
Red Beaunes France
```

```
whovian@phy504:~/host$ more reviews.txt
```

```
Reisling Terrible!
```

```
Riocha Meh
```

```
Beaunes Great!
```

```
whovian@phy504:~/host$ join -1 2 -2 1 wine.txt reviews.txt
```

```
Reisling White Germany Terrible!
```

```
Riocha Red Spain Meh
```

```
Beaunes Red France Great!
```

The **echo** command

echo displays a line of text/string passed as an argument

```
whovian@phy504:~/host$ echo 'Hello 504 Class. How are you?'
Hello 504 Class. How are you?
whovian@phy504:~/host$ echo -e '\nHello 504 Class. How are you?\n'

Hello 504 Class. How are you?

whovian@phy504:~/host$ echo -e 'Hello 504 Class. \tHow are you?'
Hello 504 Class.      How are you?
whovian@phy504:~/host$ echo -e 'Hello 504 Class. \bHow are you?'
Hello 504 Class.How are you?
whovian@phy504:~/host$ echo -e 'Hello 504 Class. \vHow are you?'
Hello 504 Class.
                How are you?
```

- The **-e** option allows the backlash to be interpreted
- The **-n** option does not skip a line after the command

Sequence

\a

\b

\c

\f

\n

\r

\t

\v

Description

Alert character, usually the ASCII BEL character.

Backspace.

Suppress the final newline in the output. Furthermore, any characters left in the argument, and any following arguments, are ignored (not printed).

Formfeed.

Newline.

Carriage return.

Horizontal tab.

Vertical tab.

The **echo** command part 2

echo displays a line of text/string passed as an argument

What if we want to redirect the output of echo to a file?

```
whovian@phy504:~/host$ echo "Hello world" >> hello.txt
whovian@phy504:~/host$ more hello.txt
Hello world
```

The **echo** outputs variables. What if we want to output the content of files? Use the command **cat**

(copy input – the file – to the output – the terminal)

```
whovian@phy504:~/host$ echo hello.txt
hello.txt
whovian@phy504:~/host$ cat hello.txt
Hello world
```

tac & **rev** commands

tac displays a line of a file in reverse ordering

rev reverses chars on each string (but not the line ordering)

```
whovian@phy504:~/host$ cat wine.txt
```

```
White Reisling Germany
```

```
Red Rioja Spain
```

```
Red Beaunes France
```

```
whovian@phy504:~/host$ tac wine.txt
```

```
Red Beaunes France
```

```
Red Rioja Spain
```

```
White Reisling Germany
```

```
whovian@phy504:~/host$ rev wine.txt
```

```
ynamreG gnilsieR etihW
```

```
niapS ahcoiR deR
```

```
ecnarF senuaeB deR
```

grep: a life saver

grep command searches lines that match a pattern

```
whovian@phy504:~$ grep "foo" <<< "foobar baz"
foobar baz
whovian@phy504:~$ grep -E "foo[a-zA-Z]+" <<< "foobar baz"
foobar baz
whovian@phy504:~$ grep "foo[a-zA-Z]+" <<< "foobar baz"
whovian@phy504:~$ grep --help
Usage: grep [OPTION]... PATTERNS [FILE]...
Search for PATTERNS in each FILE.
Example: grep -i 'hello world' menu.h main.c
PATTERNS can contain multiple patterns separated by newlines.

Pattern selection and interpretation:
-E, --extended-regexp    PATTERNS are extended regular expressions
```

grep + regular expressions = powerful tool in general
(especially useful in the wargame HW). What are
regular expressions (we will encounter them a lot!)

explain this input redirection <<< (bash)

Reasoned about input redirection (`<<<`) in Bash ^

Understanding here-strings

I'm unpacking the bash <<< operator, a here-string, which feeds a command input from a string literal and includes a trailing newline.

In Bash, the <<< operator is known as a **here-string**. It provides a convenient way to supply a string as input to a command without needing to create a separate file or use a here-document. Here's what happens and how it works:

1. String Expansion:

When you use <<< some_text , Bash first expands the string (performing any variable expansions, command substitutions, etc.).

2. Creating a Temporary Input:

The expanded string is then fed to the command as standard input. Internally, Bash creates a temporary file or uses a pipe that contains this string (with a newline appended at the end) and connects it to the command's stdin.

3. Usage Example:

For instance, consider the command:

```
bash
grep "pattern" <<< "This is some sample text containing pattern."
```

Here's what happens:

- The string "This is some sample text containing pattern." is expanded.
- The command grep "pattern" receives this string as its standard input.
- grep then searches the input for "pattern" and prints any matching lines.

4. Comparison with Other Redirections:

- Unlike input redirection from a file (using < filename), <<< allows you to quickly provide a small amount of text directly.
- It is similar in purpose to a here-document (<< EOF ... EOF), but is simpler for single-line or small inputs.

The string redirection
<<< is a good
opportunity to show how
can you use a good AI
(here, GPTO3-mini-high)
to study a language (if
you know how to ask a
question).

Warning: old models like
GPT4 not as good

Input/Output redirection

Remember command **cat** (copy input to the output).

```
whovian@phy504:~/host$ echo hello.txt  
hello.txt  
whovian@phy504:~/host$ cat hello.txt  
Hello world
```

What if we want the file to become the output (terminal input)?

> and >> symbols are two of many I/O type of redirectors

```
whovian@phy504:~/host$ cat hello.txt  
Hwhovian@phy504:~/host$ cat > hello.txt  
Hello World again with I/O redirection  
whovian@phy504:~/host$ cat hello.txt  
Hello World again with I/O redirection  
whovian@phy504:~/host$ cat >> hello.txt  
Now the new sentence will be appended to the end of the file  
whovian@phy504:~/host$ cat hello.txt  
Hello World again with I/O redirection  
Now the new sentence will be appended to the end of the file
```

**Press CTRL+D
to exit the
terminal input**

Input/Output redirection

< symbol take standard input from file

```
whovian@phy504:~/host$ read TEST < hello.txt
whovian@phy504:~/host$ echo $TEST
Hello World again with I/O redirection
```

Mixing >> and < input / output redirection

```
whovian@phy504:~/host$ more io.txt
This is IO in and out redirection (example)
whovian@phy504:~/host$ more hello.txt
Hello World again with I/O redirection
Now the new sentence will be appended to the end of the file
whovian@phy504:~/host$ cat >> hello.txt < io.txt
whovian@phy504:~/host$ more hello.txt
Hello World again with I/O redirection
Now the new sentence will be appended to the end of the file
This is IO in and out redirection (example)
```

The **curl** command

curl: command for transferring data using **PROTOCOLS**

Syntax:

curl [options] URL

Example:

```
whovian@phy504:~$ curl -o output.html https://www.geeksforgeeks.org
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	328k	100 328k	0 73156	0	0:00:04	0:00:04	--:--:-- 81317



```
whovian@phy504:~$ curl -v -o output.html https://www.geeksforgeeks.org
```

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
0	0	0	0	0	--:--:--	--:--:--	0* Host www.geeksfor

solved.

* IPv6: 2600:9000:2512:2a00:16:97f7:5900:93a1, 2600:9000:2512:8600:16:97f7:5900:93a1, 2600:9000:2512:d800:16:97f7:5900:93a1, 2600:9000:2512:3c00:16:97f7:5900:93a1, 2600:9000:2512:9200:16:97f7:5900:93a1, 2600:9000:2512:5400:16:97f7:5900:93a1

* IPv4: 18.164.124.78, 18.164.124.46, 18.164.124.35, 18.164.124.56

* Trying 18.164.124.78:443...

* Connected to www.geeksforgeeks.org (18.164.124.78) port 443

* ALPN: curl offers h2,http/1.1

} [5 bytes data]

* TLSv1.3 (OUT), TLS handshake, Client hello (1):

} [512 bytes data]

* CAfile: /etc/ssl/certs/ca-certificates.crt

The **curl** command

curl: command for transferring data using **PROTOCOLS**

Syntax:

curl [options] URL

The **file output.html** contains the **HTML** code of the webpage **geekforgeeks.org**

```
whovian@phy504:~$ more ./output.html
<!DOCTYPE html><html lang="en"><head><script type="application/ld+json">{"@context
@type":"Organization","name":"GeeksforGeeks","url":"https://www.geeksforgeeks.org/
.geeksforgeeks.org/wp-content/cdn-uploads/20200817185016/gfg_complete_logo_2x-min.
omputer science portal for geeks. It contains well written, well thought and well
nce and programming articles, quizzes and practice/competitive programming/company
"founder":[{"@type":"Person","name":"Sandeep Jain","url":"https://in.linkedin.com/
5"}],"sameAs":["https://www.facebook.com/geeksforgeeks.org/","https://twitter.com/
/www.linkedin.com/company/1299009","https://www.youtube.com/geeksforgeeksvideos/"]
econnect" href="https://fonts.googleapis.com"/><link rel="preconnect" href="https:
osorigin="true"/><meta name="viewport" content="width=device-width, initial-scale
er-scalable=1"/><title>GeeksforGeeks | A computer science portal for geeks</title>
<meta name="keywords" content="Organization, College, Companies, Institutions, Vid
gorithms, Python, Java, C, C++, JavaScript, Android Development, SQL, Data Science
, Web Development, System Design, Tutorial, Technical Blogs, Interview Experience,
Programming, Competitive Programming, SDE Sheet, Job-a-thon, Coding Contests, GAT
NodeJS, Placement, Aptitude, Quiz, Computer Science, Programming Examples, Geeks
```

The **curl** command

curl allows you to download a file (and monitor progress)

```
whovian@phy504:~$ curl --progress-bar --verbose -O http://ftp.slackware.com/pub/ftp.debian.org/debian-cdimage/12.9.0-live/amd64/iso-hybrid/debian-live-12.9.0-amd64-lxqt.iso
* Host ftp.slackware.com:80 was resolved.
* IPv6: 2600:3402:200:227::2, 2605:bc80:3010::134, 2600:3404:200:237::2
* IPv4: 140.211.166.134, 64.50.233.100, 64.50.236.52
* Trying 140.211.166.134:80...
* Connected to ftp.slackware.com (140.211.166.134) port 80
> GET /pub/ftp.debian.org/debian-cdimage/12.9.0-live/amd64/iso-hybrid/debian-live-12.9.0-amd64-lxqt.iso HTTP/1.1
> Host: ftp.slackware.com
> User-Agent: curl/8.9.1
> Accept: */*
>
#=#-#
completely sent off
< HTTP/1.1 200 OK
< Date: Thu, 30 Jan 2025 18:00:51 GMT
< Server: Apache
< Last-Modified: Sat, 11 Jan 2025 10:25:55 GMT
< ETag: "c0cfc000-62b6ba2e9f2c0"
< Accept-Ranges: bytes
< Content-Length: 3234840576
< Content-Type: application/octet-stream
<
{ [1142 bytes]
#####
8.8%AC
```

Progress %

Progress bar



--verbose allows you to download a file (and monitor progress)

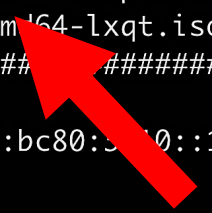
The **curl** command

curl allows you to download a file (and monitor progress)

What if the connection drops mid-download?

You can resume the transfer with **--continue-at -**

```
whovian@phy504:~$ curl --progress-bar -v -O --continue-at - http://ftp.slackware.com/pub/ftp.debian.org/debian-cdimage/12.9.0-live/amd64/iso-hybrid/debian-live-12.9.0-amd64-lxqt.iso
##### 100.0%
* Host ftp.slackware.com:80 was resolved.
* IPv6: 2600:3404:200:237::2, 2600:3402:200:227::2, 2605:bc80:3000::134
* IPv4: 64.50.233.100, 64.50.236.52, 140.211.166.134
* Trying 64.50.233.100:80...
* Connected to ftp.slackware.com (64.50.233.100) port 80
> GET /pub/ftp.debian.org/debian-cdimage/12.9.0-live/amd64/iso-hybrid/debian-live-12.9.0-amd64-lxqt.iso HTTP/1.1
> Host: ftp.slackware.com
> Range: bytes=444620800-
> User-Agent: curl/8.9.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 206 Partial Content
< Date: Thu, 30 Jan 2025 18:53:46 GMT
< Server: Apache
< Last-Modified: Sat, 11 Jan 2025 10:25:55 GMT
< ETag: "c0cfc000-62b6ba2e9f2c0"
< Accept-Ranges: bytes
< Content-Length: 2790219776
< Content-Range: bytes 444620800-3234840575/3234840576
< Content-Type: application/octet-stream
<
{ [2457 bytes data]
##### 14.5%
```



The **curl** command

What **PROTOCOLS** **curl** uses to transfer files?

```
whovian@phy504:~$ curl --help category ftp
Usage: curl [options...] <url>
auth          Authentication methods
connection    Manage connections
curl          The command line tool itself
deprecated    Legacy
dns           Names and resolving
file          FILE protocol
ftp           FTP protocol
global        Global options
http          HTTP and HTTPS protocol
imap          IMAP protocol
ldap          LDAP protocol
output        Filesystem output
pop3          POP3 protocol
post          HTTP POST specific
proxy         Options for proxies
scp           SCP protocol
sftp          SFTP protocol
smtp          SMTP protocol
ssh           SSH protocol
telnet        TELNET protocol
tftp          TFTP protocol
timeout       Timeouts and delays
tls           TLS/SSL related
upload        Upload, sending data
verbose       Tracing, logging etc
```

Some protocols, like **FTP** are not secure and serve to download public files.

Some protocols, like **SSH**, are encrypted. There is also **SFTP**, which is like **FTP** but encrypted by the **SSH** protocol.

IMAP/POP3 protocols are related to emails!

The **curl** command

Some protocols, like **FTP**, are not secure and should be used only to download public files. **curl** displays the output in the terminal (unless **-o filename** option)

```
whovian@phy504:~$ curl ftp://ftp.slackware.com/welcome.msg
```

```
YOU CAN RUN THIS (FTP URL IS AVAILABLE FOR TEACHING)
```

```
-----  
R S Y N C . O S U O S L . O R G
```

```
Oregon State University
```

```
Open Source Lab
```

```
Unauthorized use is prohibited - violators will be prosecuted  
-----
```

```
For more information about the OSL visit:
```

```
http://osuosl.org/services/hosting
```

```
This host is the home to the primary archives of several  
projects. We would prefer that only primary/secondary  
mirrors use this service. Thanks!  
-----
```

The **curl** command

In this example, I used curl with the **-o filename** option to download a preprint article from ArXiv org!

```
whovian@phy504:~$ curl -o 2501.16852 https://arxiv.org/pdf/2501.16852
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             0         0      2397k         0 --:--:-- --:--:-- --:--:-- 2419k
whovian@phy504:~$ ls
2501.16852  bash.bashrc  Dockerfile  expat.xz    host  output.html
```

curl could be used as part of a larger bash script to download entire libraries (including the whole ArXiv org). This is **dangerous**, as it would overload their servers. So, ArXiv (and many other websites) have protections against downloads by “bots” (and your script is a bot!)

The **curl** command

Another way: use capital **-O** in case you just want to download the file as it is named initially it in the server

```
whovian@phy504:~$ curl -O https://arxiv.org/pdf/2501.16852
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total  Spent    Left     Speed
100 1463k  100 1463k    0     0 3384k      0 --:--:-- --:--:-- --:--:-- 3420k
whovian@phy504:~$
```

About **overloading** their servers: if you don't need the file at this exact moment, limit the average download speed

```
whovian@phy504:~$ curl --limit-rate 100K -O https://arxiv.org/pdf/2501.16852
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total  Spent    Left     Speed
49 1463k  49  728k    0     0  101k      0 0:00:14 0:00:07 0:00:07  101k
```

The **curl** command

Another way: use capital **-O** in case you just want to download the file as it is named initially it in the server

```
whovian@phy504:~$ curl -O https://arxiv.org/pdf/2501.16852
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total   Spent    Left   Speed
100 1463k  100 1463k    0     0 3384k      0 --:--:-- --:--:-- --:--:-- 3420k
whovian@phy504:~$
```

On top of speed limits, limit the number of files per sec/minute/hour if you download multiple files.

```
whovian@phy504:~$ curl --limit-rate 2000K --rate 3/m -O https://arxiv.org/pdf/2501.168[0-9][0-9]
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total   Spent    Left   Speed
100 16.9M  100 16.9M    0     0 2015k      0 0:00:08 0:00:08 --:--:-- 2031k
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total   Spent    Left   Speed
100 363k  100 363k    0     0 2348k      0 --:--:-- --:--:-- --:--:-- 2420k
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             Dload  Upload  Total   Spent    Left   Speed
100 675k  100 675k    0     0 1997k      0 --:--:-- --:--:-- --:--:-- 2004k
```

The **curl** command

Some protocols, like **SSH/ SFTP**, are encrypted. **curl** is so versatile. Here, I display files saved on my remote server

```
whovian@phy504:~$ curl sftp://vivianmiranda@129.49.63.5:/home/vivianmiranda/extra_data/Vivian
/docker/phy504lite/
YOU CAN'T RUN THIS: YOU DON'T HAVE ACCOUNT ON MY SERVER
drwxrwxr-x   19 vivianmiranda vivianmiranda   4096 Jan  6 13:17 ..
-rw-r--r--    1 vivianmiranda vivianmiranda   1990 Jan  6 13:35 bashrc
-rw-r--r--    1 vivianmiranda vivianmiranda    959 Jan  6 14:38 jupyter_notebook_config.py
-rw-r--r--    1 vivianmiranda vivianmiranda  12147 Jan  6 15:09 Dockerfile
drwxrwxr-x    2 vivianmiranda vivianmiranda   4096 Jan  6 15:09 .
```

This example **curl (-l) sftp://...** did require me to type a passcode because I have a public-private authentication key. Once I see the file I want to download, I can use **curl** again.

```
whovian@phy504:~$ curl -o Dockerfile sftp://vivianmiranda@129.49.63.5:/home/vivianmiranda/ext
ra_data/Vivian/docker/phy504lite/Dockerfile
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %    12147    0      0  10662      0  0:00:01  0:00:01  --:--:-- 10692
100 12147  100 12147    0     0  10659      0  0:00:01  0:00:01  --:--:-- 10659
```

The **curl** command

curl is massive. There is an online book (pdf – 500 pages) about just this command

Everything curl <https://everything.curl.dev>

Everything curl is an extensive guide for all things curl. The project itself, the command-line tool, the library, how everything started and how it came to be the useful tool it is today. It explains how we work on developing it further, what it takes to use it, how you can contribute with code or bug reports and why millions of existing users use it.

This book is meant to be interesting and useful to both casual readers and somewhat more experienced developers. It offers something for everyone to pick and choose from.

Do not read this book from front to back. Read the chapters or content you are curious about and flip back and forth as you see fit.

This book is an open source project in itself: open, completely free to download and read. Free for anyone to comment on, and available for everyone to contribute to and help out with. Send your bug reports, ideas, pull requests or critiques to us and I or someone else will work on improving the book accordingly.

This book will never be finished. I intend to keep working on it. While I may at some point consider it fairly complete, covering most aspects of the project (even if only that seems like an insurmountable goal), the curl project will continue to move so there will always be things to update in the book as well.

This book project started at the end of September 2015.

Find files with **find** command

find: search for files and directories within a directory hierarchy

Syntax:

find [path] [expression]

There are many powerful options associated w/ **find**

```
whovian@phy504:~$ find /home/whovian/ -maxdepth 1 -type d
/home/whovian/
/home/whovian/.ssh
/home/whovian/.emacs.d
/home/whovian/test
/home/whovian/.local
/home/whovian/host
/home/whovian/.jupyter
/home/whovian/.cmake
```



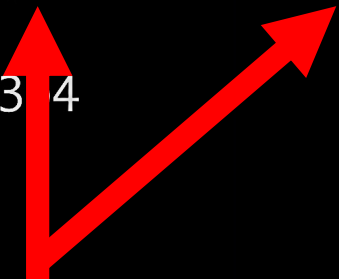
Only child of
home directory

directory

Find files with **find** command

find: search for files and directories within a directory hierarchy

```
whovian@phy504:~$ find /home/whovian/ -mindepth 2 -maxdepth 3 -type d
/home/whovian/.emacs.d/elc-cache
/home/whovian/.emacs.d/elc-cache/29.4-fb066314
/home/whovian/.local/share
/home/whovian/.local/share/nano
/home/whovian/host/old_notes
/home/whovian/host/Lecture2
/home/whovian/host/Lecture5
/home/whovian/host/Lecture4
/home/whovian/host/Lecture3
/home/whovian/host/Lecture6
/home/whovian/host/Lecture1
/home/whovian/host/oldDocker
/home/whovian/host/oldDocker/teste_hidden
/home/whovian/host/oldDocker/CPP
/home/whovian/host/oldDocker/exam
/home/whovian/host/oldDocker/C
/home/whovian/.cmake/packages
/home/whovian/.cmake/packages/armadillo
whovian@phy504:~$
```



Only grandchild and
grand grandchild of the
home directory

What if you
don't want to
show hidden
folders?

Find files with **find** command

find: search for files and directories within a directory hierarchy

```
whovian@phy504:~$ find /home/whovian/ -mindepth 2 -maxdepth 3 -type d -not -path "*/.*"
/home/whovian/host/old_notes
/home/whovian/host/Lecture2
/home/whovian/host/Lecture5
/home/whovian/host/Lecture4
/home/whovian/host/Lecture3
/home/whovian/host/Lecture6
/home/whovian/host/Lecture1
/home/whovian/host/oldDocker
/home/whovian/host/oldDocker/teste_hidden
/home/whovian/host/oldDocker/CPP
/home/whovian/host/oldDocker/exam
/home/whovian/host/oldDocker/C
whovian@phy504:~$
```

What if you
don't want to
show hidden
folders?

```
whovian@phy504:~$ find /home/whovian/ -mindepth 2 -maxdepth 3 -type d -path "*/.*"
/home/whovian/.emacs.d/elc-cache
/home/whovian/.emacs.d/elc-cache/29.4-fb0663b4
/home/whovian/.local/share
/home/whovian/.local/share/nano
/home/whovian/.cmake/packages
/home/whovian/.cmake/packages/armadillo
```

Only hidden
folders

Find files with **find** command

find: search files with different types

```
whovian@phy504:~$ touch vivian.jpg
whovian@phy504:~$ touch miranda.png
whovian@phy504:~$ find /home/whovian/ \( -name "*.jpg" -or -name "*.png" \)
/home/whovian/vivian.jpg
/home/whovian/miranda.png
whovian@phy504:~$
```



Logical operator

find has the following operators listed in order of precedence (highest -> lowest)

1. **()**
2. **!|-not**
3. **-a|-and**
4. **-o|-or**
5. **,** (GNU only)

Note: All **tests** and **actions** have an implied **-a** linking each other

\() is not necessary in this case (just **-or** operators). If you mix the **-and** with **-or**, **\()** may be needed



```
whovian@phy504:~$ find . \( -name "*.jpg" -or -name "*.png" \) -or \( -name "elisa*" \)
./elisa.abc
./vivian.jpg
./miranda.png
whovian@phy504:~$
```

Find files with **find** command

find: very important to carefully consider operator precedence

```
whovian@phy504:~$ ls
bash.bashrc elisa.abc host miranda.png test vivian.jpg
whovian@phy504:~$ find . -type f \( \( -name "*.jpg" -or -name "*.png" \) -or -name "elisa*" \) -exec gzip {} \;
whovian@phy504:~$ ls
bash.bashrc elisa.abc.gz host miranda.png.gz test vivian.jpg.gz
```

The double **\(\(... \) \) -exec gzip {} \;** allows the gzip to apply to all matched files. Without the second **\(\)**, the command **-exec gzip {} \;** only apply to elisa.abc

```
whovian@phy504:~$ ls
bash.bashrc elisa.abc host miranda.png test vivian.jpg
whovian@phy504:~$ find . -type f \( -name "*.jpg" -or -name "*.png" \) -or -name "elisa*" -exec gzip {} \;
whovian@phy504:~$ ls
bash.bashrc elisa.abc.gz host miranda.png test vivian.jpg
whovian@phy504:~$
```

What This Means

ChatGPT o3-mini-high got it right

- Files like **vivian.jpg** and **miranda.png** satisfy the first part (they are regular files matching *.jpg or *.png). However, there is **no -exec action attached to that branch**.
- Only files matching **elisa*** (like **elisa.abc**) fall into the second branch where the **-exec gzip {} ;** is attached. Therefore, only **elisa.abc** gets gzipped.

Find files with **find** command

Operator “\;” after **--exec** implies command applies 1x per file

What if we want to compress all files to a single file?

```
whovian@phy504:~$ find . -type f \( \( -name "*.jpg" -or -name "*.png" \) -or \( -name "elisa*" \) \) -print0 | tar --null -cvzf archive.tar.gz --files-from=-
./elisa.abc
./vivian.jpg
./miranda.png
whovian@phy504:~$
whovian@phy504:~$ ls
archive.tar.gz  bash.bashrc  elisa.abc  host  miranda.png  test  vivian.jpg
```

- **-print0**

This makes **find** output file names followed by a null character (instead of a newline), which is safer for file names with spaces or unusual characters.

- **tar --null -cvzf archive.tar.gz --files-from=-**
 - **--null** tells **tar** to expect null-separated file names.
 - **-c** creates a new archive.
 - **-v** makes **tar** list the files being processed (verbose mode).
 - **-z** compresses the archive using **gzip**.
 - **-f archive.tar.gz** specifies the output file name.
 - **--files-from=-** tells **tar** to read the list of file names from standard input (provided by **find**).
↓

HW1: create examples that...

- extract the second and third field of a space-separated file
- extract the first 10 characters of each line in a file
- extract the characters from the 11th to the 20th of each line in a file
- extract the last field of a file using the delimiter ":" (the file `/etc/passwd` is an example of such a file)
- extract the first 3 characters of each line in a file, starting from the 2nd character
- expand the `wine.txt` and `review.txt` files – include some unpaired wines in both files (wines that only exist in one file). Then:
 - write an example that joins the files and only displays the unique lines
 - write an example that joins the files and displays paired lines + unpaired lines from the first file
 - write a command that joins two files based on the first field assuming comma-separated fields

HW2: create examples that...

- Assume you have a CSV file named `data.csv` where each line has the format: “`username,email,age,country`”. Extract and display only the **email** from each line.
- Suppose you have a file `records.txt` where each line is formatted as follows: `id:username:score`. Write a command that uses:
 - The command `grep` to find the record for the user bob (case-sensitive),
 - The command `cut` to extract the `score`
 - The command `rev` to reverse the digits of the score.
- (!!) A file named `info.txt` contains precisely one line formatted as follows: “`key1:value1:key2:value2:key3:value3`”. Write a Bash script that uses `cut` to extract the value corresponding to `key2`, store it in a variable called `value2`, and then print “`The value of key2 is: value2`”. Tip. To read the file, you will need some concepts from the next class. The command should be something like `line=$(cat info.txt)`
- (!!) A file named `data.csv` contains lines with exactly five comma-separated fields. Write a Bash command that extracts the 5th field from every line, reverses the content of that field, and writes the results to `field5_reversed.txt`.

HW3: create examples that...

- Print a simple message to the console:
- Print the current date (hint: use the date command)
- Append a custom sentence to a file
- Print the contents of a file (hint: you need both **echo** and **cat**)
- Print the current working directory
- Print multiple lines using the **-e** option
- Display the contents of a file on the terminal
- Concatenate the contents of two or more files and print them to the terminal
- Create a new file and add content without an editor (nano, emacs...). Hint: use **cat**
- Reverse the lines of a file and write the output to a new file
- Reverse the lines of multiple files

HW4: **cat** is a powerful command

- **VMB** are the initials of my name. Can you explain didactically what the **VMB** initials are doing in the cat command below? (you can run the program yourself to check)

```
whovian@phy504:~$ cat << 'EOF' > script.sh
#!/bin/bash
echo -n "Hello World"
EOF
whovian@phy504:~$ cat << 'VMB' > script2.sh
#!/bin/bash
echo -n "Hello World"
VMB
```

- Do you think the **EOF** and **VMB** initials will be printed to the files **script.sh** and **script2.sh**? Check your answer!

HW5: **cat** is a powerful command

- We have not studied the **bash** commands **sort** and **uniq**. But we all know English, and **bash** is somewhat intuitive. Then, could you guess what the output of the command will be?

cat script.sh script2.sh | sort | uniq -u

```
whovian@phy504:~$ cat << 'OMG' > script.sh
```

```
> #!/bin/bash
```

```
echo -n "Hello World"
```

```
echo -n "ops, I missed \n"
```

```
echo -n "Are you having fun Phy504?"
```

```
OMG
```

```
whovian@phy504:~$ cat << 'VMB' > script2.sh
```

```
> #!/bin/bash
```

```
echo -n "Hello World"
```

```
echo -n "No, I am not having fun! Just kidding"
```

```
echo -n "Phy504 is awesome"
```

```
VMB
```

```
whovian@phy504:~$ cat script.sh script2.sh | sort | uniq -u
```

HW6: working with **curl**

- The **speed.hetzner.com** website provides some large files that allow users to stress test their internet connection. One of these files is 1GB in size and is available at (<https://fsn1-speed.hetzner.com/1GB.bin>).
 - **(easy)** Use the file to reproduce some **curl** examples (related to the download of files) we showed in this lecture.
- (!!!) In the following few lectures, we will see how to run a while loop in bash. We will also see how to create variables inside a script. But here, I want to challenge you to see if you can figure out some of these concepts yourself. If not, wait a few lectures to do this HW. The task is to *create a script using **while loops** that retry downloads on failure up to some maximum number of attempts.*
- (!!) While loops are ugly (low-level programming)! Use the **man** command or **--help** option to check if we can accomplish a similar goal (retry download up to max attempts) using **curl** options.

HW7: working with **curl** x2

- The **diff** bash command analyzes the difference between two files:

```
whovian@phy504:~$ cat > test1.txt
This is file number one (ONE, 1)
^X
whovian@phy504:~$
whovian@phy504:~$ cat > test2.txt
This is file number two (TWO, 2)whovian@phy504:~$
whovian@phy504:~$
whovian@phy504:~$ diff test1.txt test2.txt
1,2c1
< This is file number one (ONE, 1)
<
---
> This is file number two (TWO, 2)
\ No newline at end of file
```

Download two large files with the **curl** command and show only the first 10 lines of their differences.

- Process substitution is a fantastic **bash** feature. It allows the user to redirect the output of multiple commands as input to another command. How does it work? With **diff**, it is more or less like:

diff < (command 1) < (command 2)

Redo the previous exercise using process substitution. Did you notice anything related to the speed of the overall script?