Interacting with Programs

Linux Command Line

Yan Shoshitaishvili
Arizona State University

## Hello Students

You've (better have) seen this:

```
# ./helloworld
Hello world!
```

But what happens under the hood?

Linux

In this class, we'll focus on Linux.

Linux facilitates the safe interaction of **Processes** with each other and with the **File System**, the **Network**, and **Computer Hardware**.

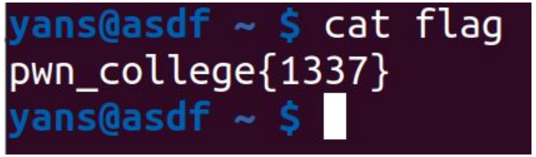
The pwn.college infrastructure provides you a with a Linux environment. You will interact with this environment via the **Command Line**.

Command Line?

The command line (aka "shell") is a powerful interface to a computer. Basic idea:

1. You type a command.
2. The system executes it and outputs the results.

Typically, a command will contain a program name and arguments to that program, separated by spaces.



```
yans@asdf ~ $ cat flag
pwn_college{1337}
yans@asdf ~ $ 
```

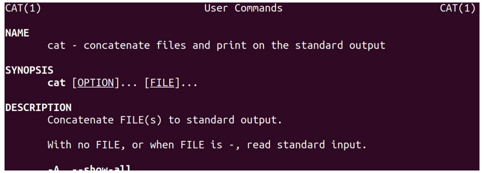
What happened?

1. I told the shell to run the program `cat` with the argument `flag`.
2. The shell found the `cat` program file and launched it into a cat `process` with a `flag` argument.
3. `cat` is a program that outputs files. It reads the `flag` argument and knows to output the `flag` file, which contains "pwn_college{1337}".

# Learning the Command Line

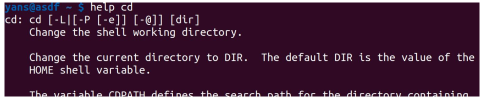
You **really** should be comfortable with the command line by now. If you are not:

1. Bandit (https://overthewire.org/wargames/bandit/) is a hands-on "wargame" that will teach you how to use the commandline.
2. There are many online tutorials.
3. Documentation!
   a. **man** (manual) pages. Example of `man cat`:



```
CAT(1)                           User Commands                          CAT(1)

NAME
       cat - concatenate files and print on the standard output

SYNOPSIS
       cat [OPTION]... [FILE]...

DESCRIPTION
       Concatenate FILE(s) to standard output.

       With no FILE, or when FILE is -, read standard input.

       -A, --show-all
```

   b. **help** for shell "builtins". For example, `help cd`:



```
yans@asdf ~ $ help cd
cd: cd [-L|[-P [-e]] [-@]] [dir]
    Change the shell working directory.

    Change the current directory to DIR.  The default DIR is the value of the
    HOME shell variable.

    The variable CDPATH defines the search path for the directory containing
```

4. Ask for help on discord.

Processes

A process is a running program.

A program is a file on your computer.

Files live in a filesystem.

Your web browser, your command line interpreter ("shell"), your text editor, all start out as files on the filesystem and become processes when they are executed.

We'll learn more about processes in the rest of this module.

The File System

Files are organized into file systems.

Unlike Windows (which traditionally has different file systems at different anchor points `C:\`, `D:\`, `E:\`, etc.), Linux presents a unified file system view.

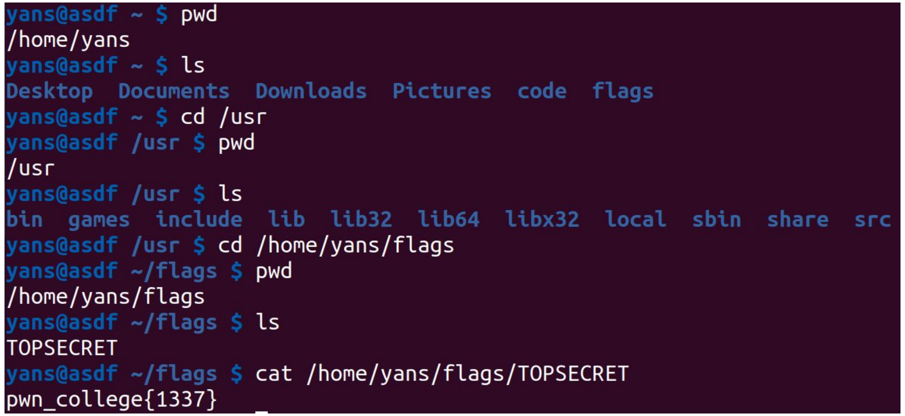
`/`	The "anchor" of the filesystem. Pronounced "root".
`/usr`	The **U**nix **S**ystem **R**esource. Contains all the system files.
`/usr/bin`	Executable files for programs installed on the computer.
`/usr/lib`	Shared libraries for use by programs on the computer.
`/usr/share`	Program resources (icons, art assets, etc).
`/etc`	System configuration.
`/var`	Logs, caches, etc.
`/home`	User-owned data.
`/home/hacker`	Data owned by you in the pwn.college infrastructure.
`/proc`	Runtime process data.
`/tmp`	Temporary data storage.

Directories

Files are stored in **directories** in the filesystem. Each directory has several files.

Each process has a "current working directory". You can view it with the `pwd` builtin (and it usually shows in your prompt) and change it with the `cd` builtin.

You can *list* the files in a directory using the `ls` command. With no arguments, it will list the files in the current directory.



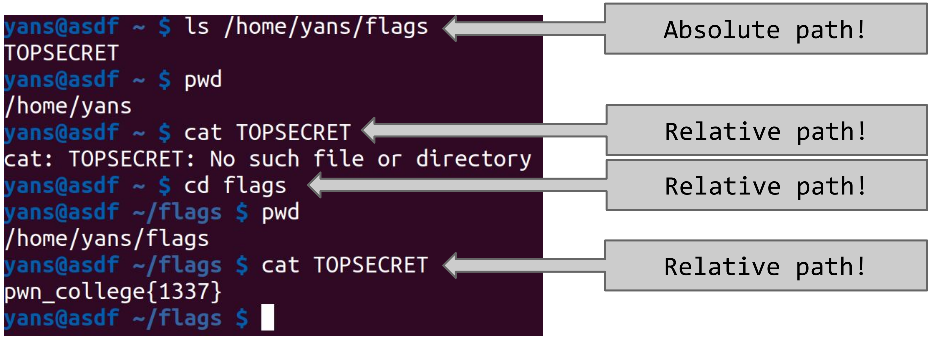
```
yans@asdf ~ $ pwd
/home/yans
yans@asdf ~ $ ls
Desktop  Documents  Downloads  Pictures  code  flags
yans@asdf ~ $ cd /usr
yans@asdf /usr $ pwd
/usr
yans@asdf /usr $ ls
bin  games  include  lib  lib32  lib64  libx32  local  sbin  share  src
yans@asdf /usr $ cd /home/yans/flags
yans@asdf ~/flags $ pwd
/home/yans/flags
yans@asdf ~/flags $ ls
TOPSECRET
yans@asdf ~/flags $ cat /home/yans/flags/TOPSECRET
pwn_college{1337}
```

Specifying paths...

There are two ways to specify paths:

Absolute Paths start with `/`, such as `/usr`, `/home/yans/flags/TOPSECRET`, etc.

Relative Paths *don't* start with `/`, and are relative to the current working directory.

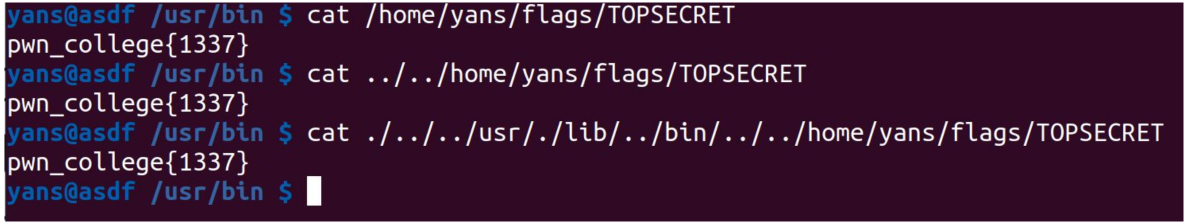


```
yans@asdf ~ $ ls /home/yans/flags
TOPSECRET
yans@asdf ~ $ pwd
/home/yans
yans@asdf ~ $ cat TOPSECRET
cat: TOPSECRET: No such file or directory
yans@asdf ~ $ cd flags
yans@asdf ~/flags $ pwd
/home/yans/flags
yans@asdf ~/flags $ cat TOPSECRET
pwn_college{1337}
yans@asdf ~/flags $
```

# Closer look: Paths

A "path" contains:

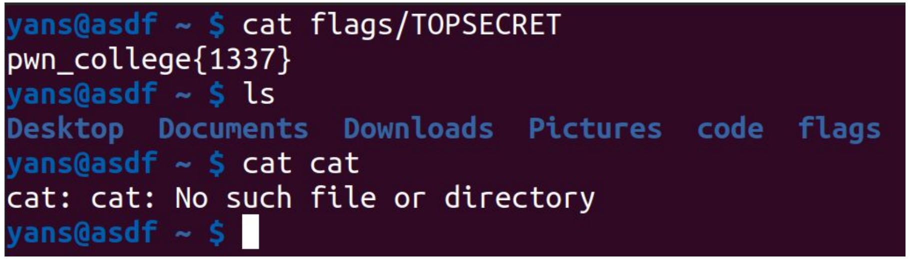
- Possible leading "/" to specify that the path is absolute (starts at the root).
- Directory names, followed by "/" to reference resources "inside" a directory.
- A ".", signifying "current directory".
- A "..", signifying "the directory that the current directory lives in".
- A file name at the end of the path, referencing a file with that name.



```
yans@asdf /usr/bin $ cat /home/yans/flags/TOPSECRET
pwn_college{1337}
yans@asdf /usr/bin $ cat ../../home/yans/flags/TOPSECRET
pwn_college{1337}
yans@asdf /usr/bin $ cat ./../../usr/./lib/../bin/../../home/yans/flags/TOPSECRET
pwn_college{1337}
yans@asdf /usr/bin $ 
```

Paths to commands

Wait a second... Where is `cat`???



```
yans@asdf ~ $ cat flags/TOPSECRET
pwn_college{1337}
yans@asdf ~ $ ls
Desktop  Documents  Downloads  Pictures  code  flags
yans@asdf ~ $ cat cat
cat: cat: No such file or directory
yans@asdf ~ $ 
```

If the first word of the command has no `/` characters, the shell will search for it in either its builtins or a set of directories specified in the `PATH` environment variable.

What?

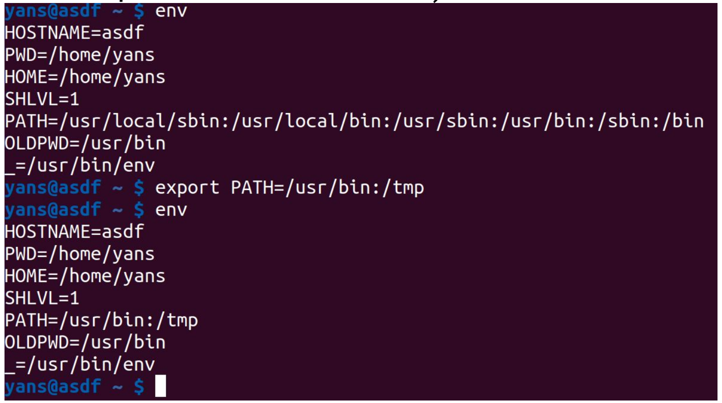
Interlude: Environment Variables?

"Environment variables" are a set of Key/Value pairs passed into every process when it is launched. Critical variables:

> `PATH`: a list of directories to search for programs in.
> `PWD`: the current working directory (same as the pwd command)
> `HOME`: the path to your home directory
> `HOSTNAME`: the name of your system

You can print environment variables with the `env` command, and set them with shell syntax.

`env` is a very useful command. Study its man page!

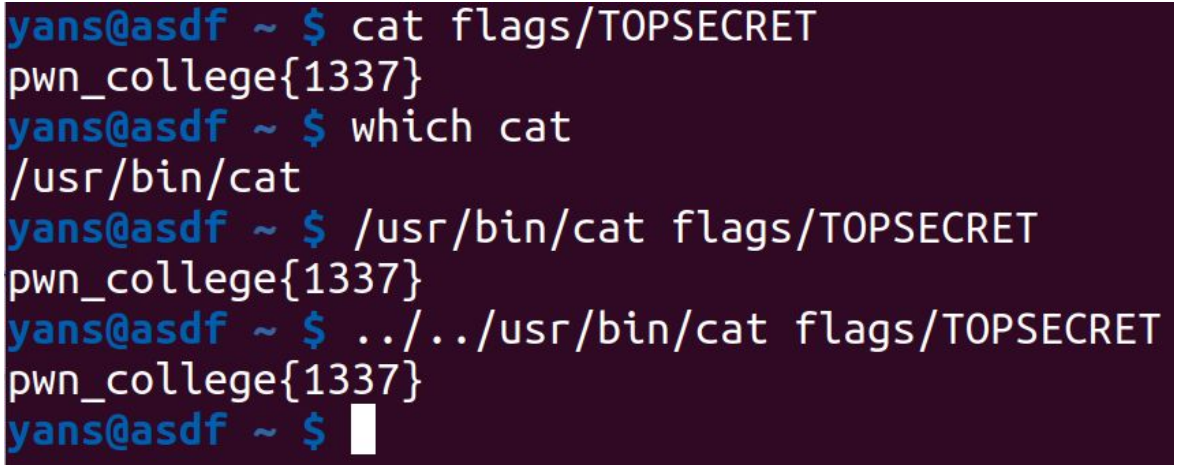


```
yans@asdf ~ $ env
HOSTNAME=asdf
PWD=/home/yans
HOME=/home/yans
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
OLDPWD=/usr/bin
_=/usr/bin/env
yans@asdf ~ $ export PATH=/usr/bin:/tmp
yans@asdf ~ $ env
HOSTNAME=asdf
PWD=/home/yans
HOME=/home/yans
SHLVL=1
PATH=/usr/bin:/tmp
OLDPWD=/usr/bin
_=/usr/bin/env
yans@asdf ~ $ 
```

Back to paths.

If you're curious about what program file ends up becoming your `cat` process after it's found using the `PATH` variable, use `which`.

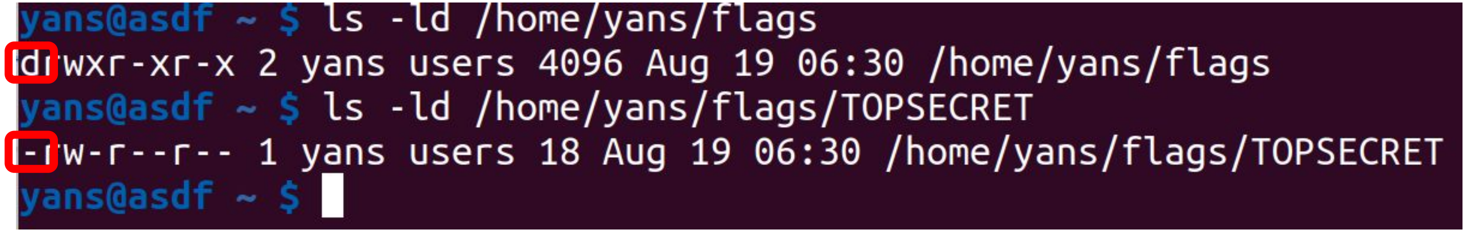
You can also launch programs with absolute or relative paths, which will not rely on `PATH`.



```
yans@asdf ~ $ cat flags/TOPSECRET
pwn_college{1337}
yans@asdf ~ $ which cat
/usr/bin/cat
yans@asdf ~ $ /usr/bin/cat flags/TOPSECRET
pwn_college{1337}
yans@asdf ~ $ ../../usr/bin/cat flags/TOPSECRET
pwn_college{1337}
yans@asdf ~ $ 
```

A deeper look at files

There are many different types of files. You can use `ls -ld /path/to/your/file` to check.



```
yans@asdf ~ $ ls -ld /home/yans/flags
drwxr-xr-x 2 yans users 4096 Aug 19 06:30 /home/yans/flags
yans@asdf ~ $ ls -ld /home/yans/flags/TOPSECRET
-rw-r--r-- 1 yans users 18 Aug 19 06:30 /home/yans/flags/TOPSECRET
yans@asdf ~ $ 
```

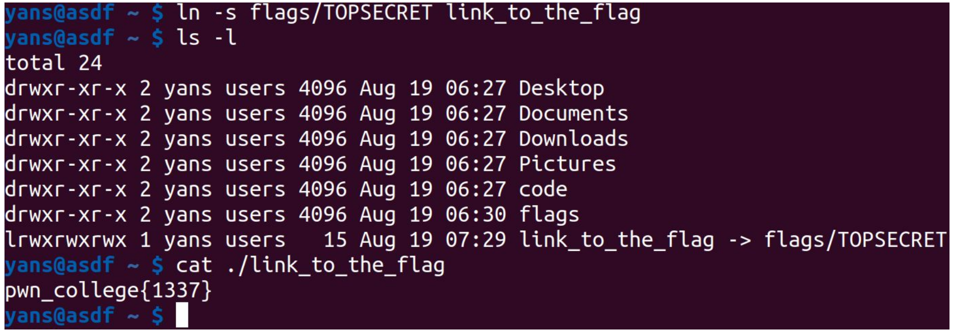
Types:

`-` is a **regular file**
`d` is a **directory** (yes, directories are actually just special files!)
`l` is a **symbolic link** (a file that transparently points to another file or directory)
`p` is a **named pipe** (also known as a FIFO. You will get very familiar with these this module!)
`c` is a **character device file** (i.e., backed by a hardware device that produces or receives data streams, such as a microphone)
`b` is a **block device file** (i.e., backed by a hardware device that stores and loads blocks of data, such as a hard drive)
`s` is a **unix socket** (essentially a local network connection encapsulated in a file)

Symbolic (AKA soft) links?

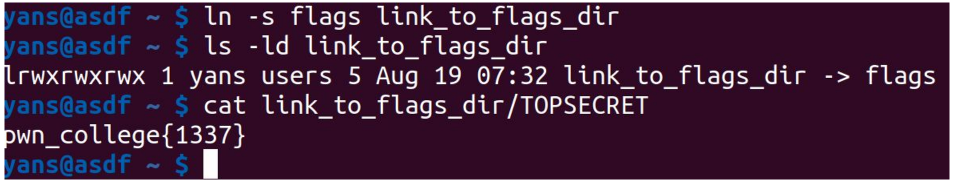
A symbolic/soft link is a special type of file that references another file.

They are created `ln -s` (`-s` stands for symbolic. Read the man page!).



```
yans@asdf ~ $ ln -s flags/TOPSECRET link_to_the_flag
yans@asdf ~ $ ls -l
total 24
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Desktop
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Documents
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Downloads
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Pictures
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 code
drwxr-xr-x 2 yans users 4096 Aug 19 06:30 flags
lrwxrwxrwx 1 yans users   15 Aug 19 07:29 link_to_the_flag -> flags/TOPSECRET
yans@asdf ~ $ cat ./link_to_the_flag
pwn_college{1337}
yans@asdf ~ $
```

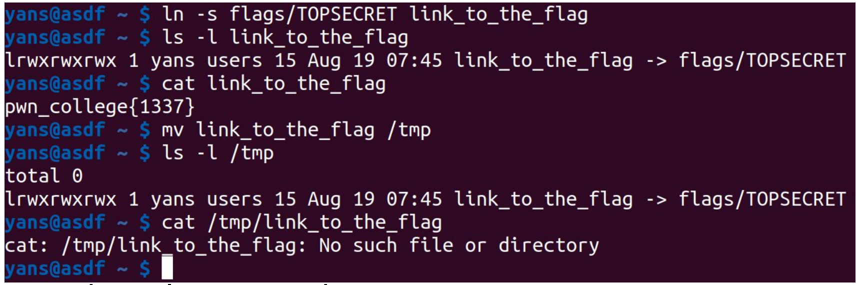
You can also link directories:



```
yans@asdf ~ $ ln -s flags link_to_flags_dir
yans@asdf ~ $ ls -ld link_to_flags_dir
lrwxrwxrwx 1 yans users 5 Aug 19 07:32 link_to_flags_dir -> flags
yans@asdf ~ $ cat link_to_flags_dir/TOPSECRET
pwn_college{1337}
yans@asdf ~ $
```

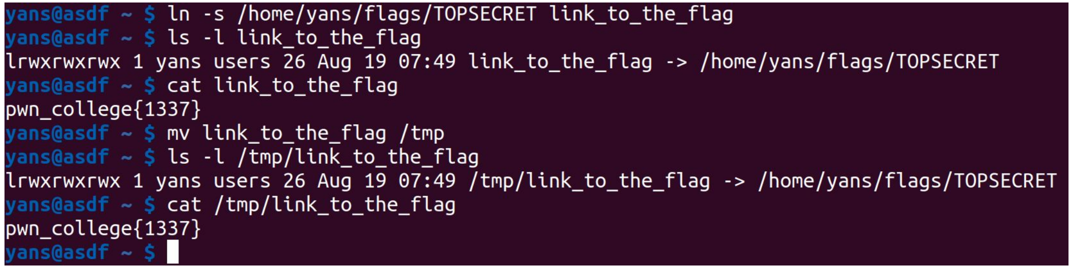
Symbolic link gotchas

Beware: symbolic links to relative paths are relative to the directory containing the link!



```
yans@asdf ~ $ ln -s flags/TOPSECRET link_to_the_flag
yans@asdf ~ $ ls -l link_to_the_flag
lrwxrwxrwx 1 yans users 15 Aug 19 07:45 link_to_the_flag -> flags/TOPSECRET
yans@asdf ~ $ cat link_to_the_flag
pwn_college{1337}
yans@asdf ~ $ mv link_to_the_flag /tmp
yans@asdf ~ $ ls -l /tmp
total 0
lrwxrwxrwx 1 yans users 15 Aug 19 07:45 link_to_the_flag -> flags/TOPSECRET
yans@asdf ~ $ cat /tmp/link_to_the_flag
cat: /tmp/link_to_the_flag: No such file or directory
yans@asdf ~ $
```

vs absolute path:

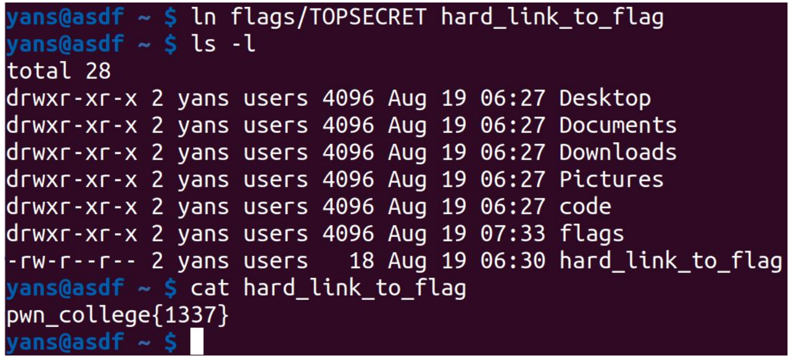


```
yans@asdf ~ $ ln -s /home/yans/flags/TOPSECRET link_to_the_flag
yans@asdf ~ $ ls -l link_to_the_flag
lrwxrwxrwx 1 yans users 26 Aug 19 07:49 link_to_the_flag -> /home/yans/flags/TOPSECRET
yans@asdf ~ $ cat link_to_the_flag
pwn_college{1337}
yans@asdf ~ $ mv link_to_the_flag /tmp
yans@asdf ~ $ ls -l /tmp/link_to_the_flag
lrwxrwxrwx 1 yans users 26 Aug 19 07:49 /tmp/link_to_the_flag -> /home/yans/flags/TOPSECRET
yans@asdf ~ $ cat /tmp/link_to_the_flag
pwn_college{1337}
yans@asdf ~ $
```

Hard links?

The existence of soft links implies a hard link.

Hard links (created with `ln` without the `-s` argument) reference the original file directly by performing magic with scary words such as "inode".



```
yans@asdf ~ $ ln flags/TOPSECRET hard_link_to_flag
yans@asdf ~ $ ls -l
total 28
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Desktop
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Documents
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Downloads
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 Pictures
drwxr-xr-x 2 yans users 4096 Aug 19 06:27 code
drwxr-xr-x 2 yans users 4096 Aug 19 07:33 flags
-rw-r--r-- 2 yans users   18 Aug 19 06:30 hard_link_to_flag
yans@asdf ~ $ cat hard_link_to_flag
pwn_college{1337}
yans@asdf ~ $ 
```

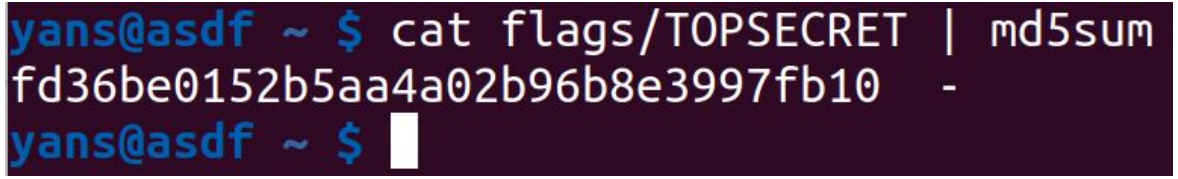
A hard link is an equally "valid" reference to the original file as the original file itself. It is a file that happens to be backed by the same data as the original.

Further reading: https://medium.com/@meghamohan/hard-link-and-symbolic-link-3cad74e5b5dc

Pipes!

Pipes facilitate a unidirectional flow of information. There are two types of pipes:

1. Unnamed pipes, ethereal channels of information between processes. Most commonly used to direct data from one command to another.



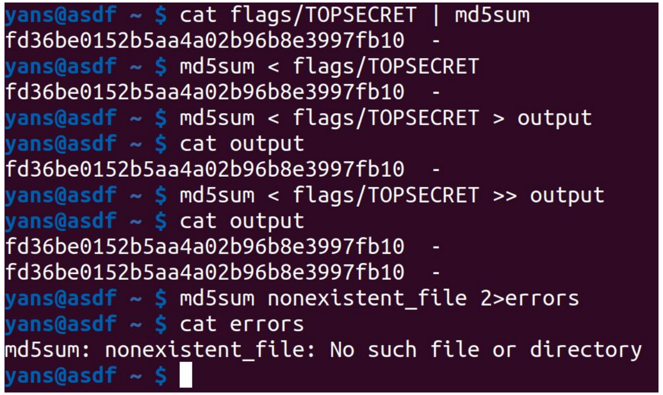
```
yans@asdf ~ $ cat flags/TOPSECRET | md5sum
fd36be0152b5aa4a02b96b8e3997fb10  -
yans@asdf ~ $ 
```

2. Named pipes, also known as FIFOs, created using the "mkfifo" command. Also used to help facilitate data flow in certain situations.

Input and output redirection

Command output can be redirected to files, and command input can be provided from files.

`<in_file:`	redirect in_file into the command's input
`>out_file:`	redirect the command's output into out_file, overwriting it
`>>out_file:`	redirect the command's output into out_file, appending to it
`2>error_file:`	redirect the command's errors into error_file, overwriting it
`2>>error_file:`	redirect the command's errors into error_file, appending to it



```
yans@asdf ~ $ cat flags/TOPSECRET | md5sum
fd36be0152b5aa4a02b96b8e3997fb10  -
yans@asdf ~ $ md5sum < flags/TOPSECRET
fd36be0152b5aa4a02b96b8e3997fb10  -
yans@asdf ~ $ md5sum < flags/TOPSECRET > output
yans@asdf ~ $ cat output
fd36be0152b5aa4a02b96b8e3997fb10  -
yans@asdf ~ $ md5sum < flags/TOPSECRET >> output
yans@asdf ~ $ cat output
fd36be0152b5aa4a02b96b8e3997fb10  -
fd36be0152b5aa4a02b96b8e3997fb10  -
yans@asdf ~ $ md5sum nonexistent_file 2>errors
yans@asdf ~ $ cat errors
md5sum: nonexistent_file: No such file or directory
yans@asdf ~ $ 
```