Quick Announcements

Update on Case Study

- We are past the extended deadline now…

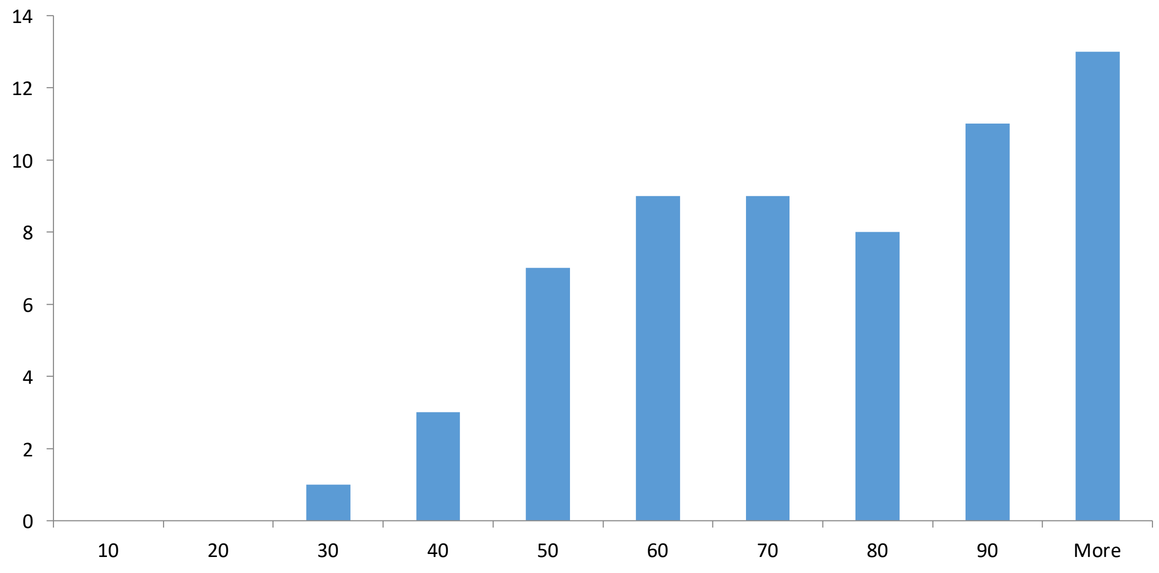
Update on upcoming IE2

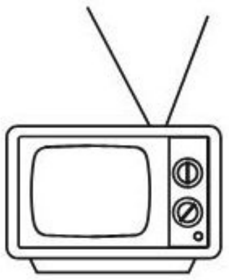
- In classroom as before
- schedule your accommodations ahead of time
- Focus on modules M1 + M2

Quick Announcements

Update on GQ02

- Curving applied





Previously On... CIS4930

- Redirections Rudiments
  - To / from files
  - Appending / overwriting
- Merging streams

New syntax to merge 2 streams

- New Syntax!!!

2>&1

"redirecting FD#2 to where FD#1 is pointing at right now"

- New bug!!!! ;p

```
./myprog.sh 2>&1
This is something for STDOUT
This is something for STDERR

# ok but how do I check these were both
# sent to STDOUT?
```

New syntax to merge 2 streams

- New Syntax!!!

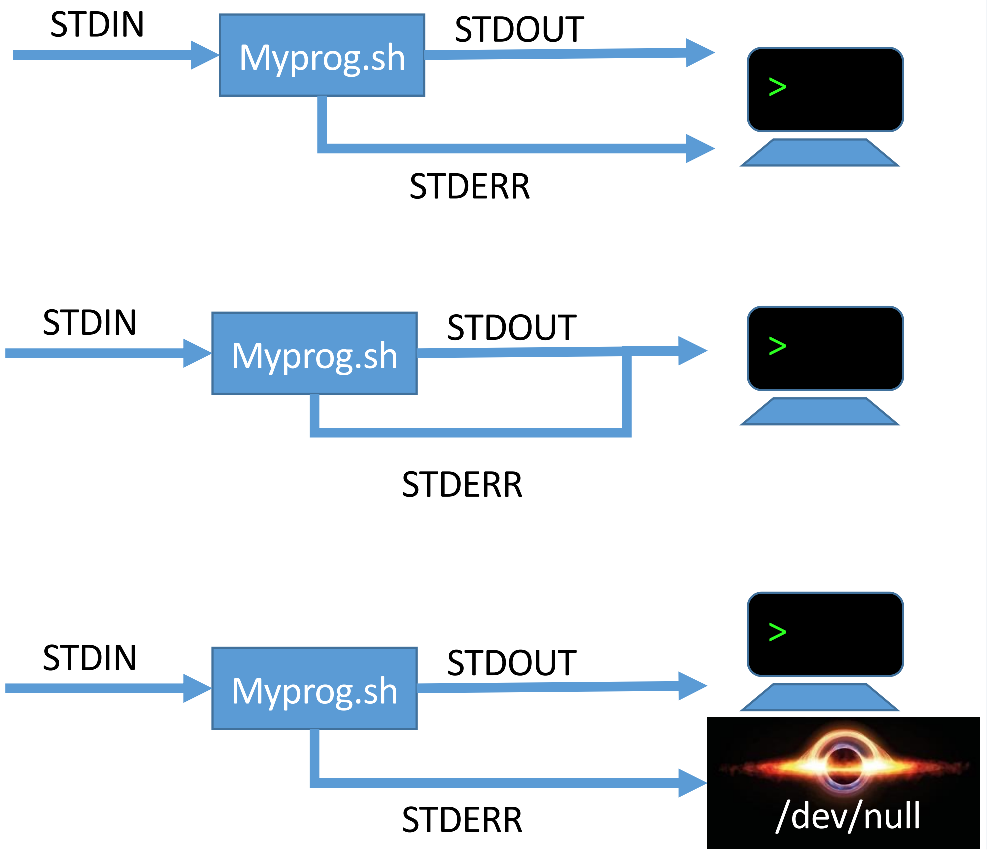
2>&1

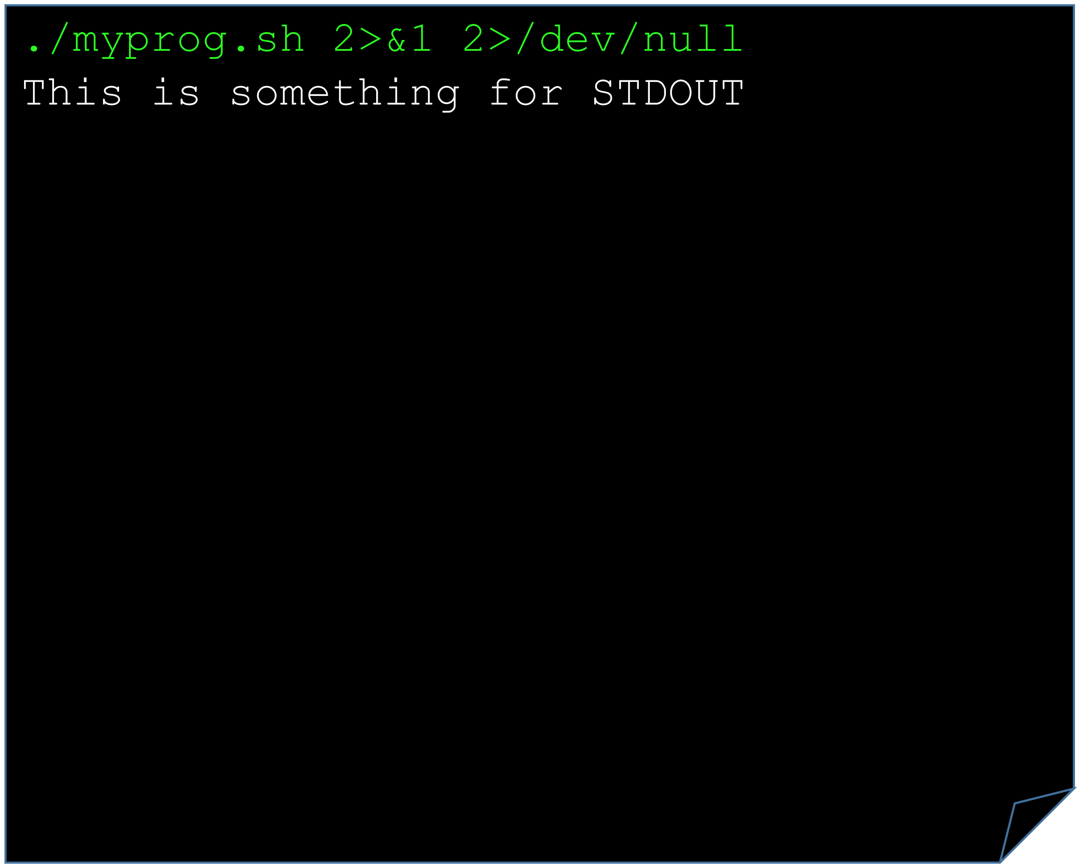
"redirecting FD#2 to where FD#1 is pointing at right now"

- New bug!!!! ;p

```
./myprog.sh 2>&1
This is something for STDOUT
This is something for STDERR

# ok but how do I check these were both
# sent to STDOUT?

# Redirecting 2> to null should have no
# effects since it is already on 1

./myprog.sh 2>&1 2>/dev/null
This is something for STDOUT

# WAIT!? WHAT????
# FD#2 was set to FD#1 destination
# but then we reset FD#2 to /dev/null
```

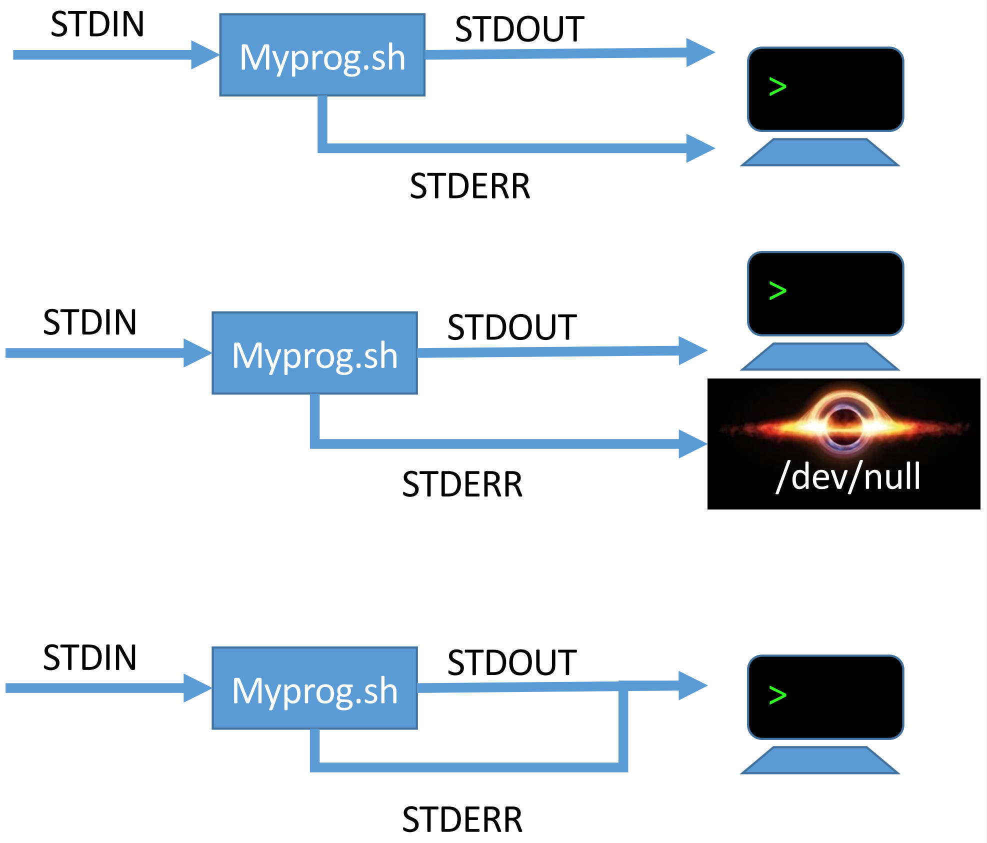
So, what happened exactly?

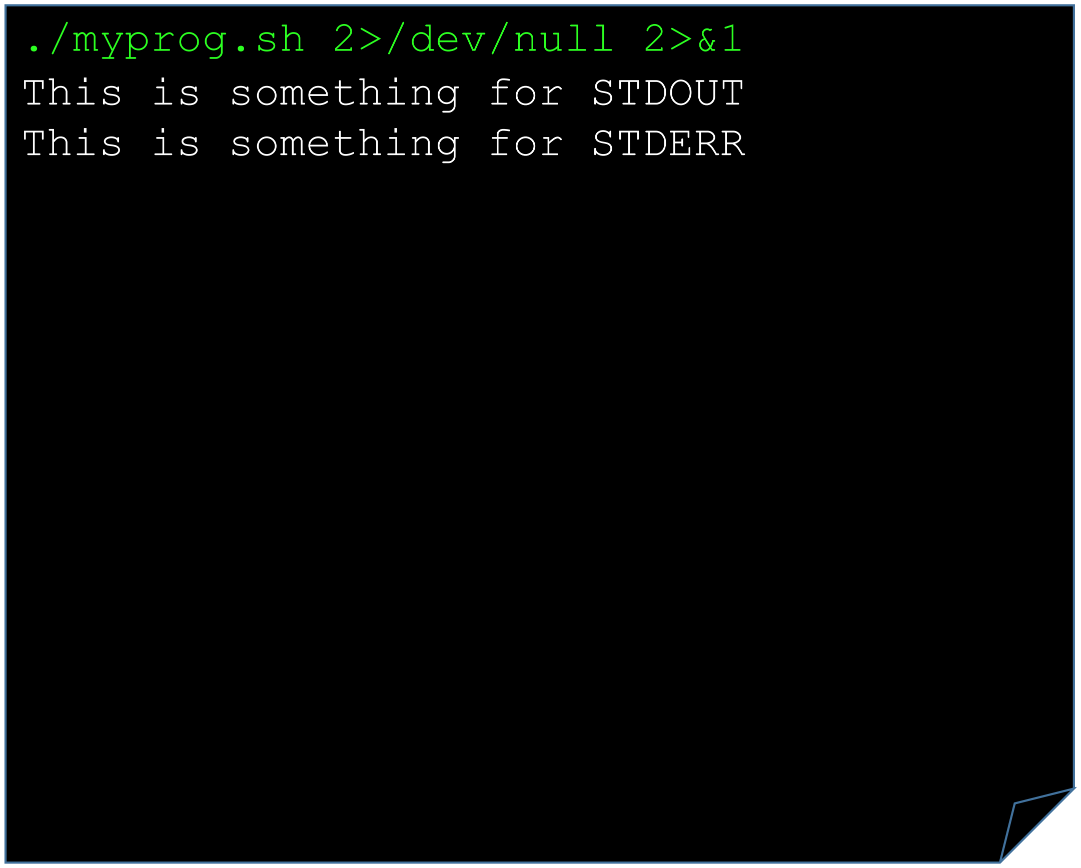




```
./myprog.sh 2>&1 2>/dev/null
This is something for STDOUT
```

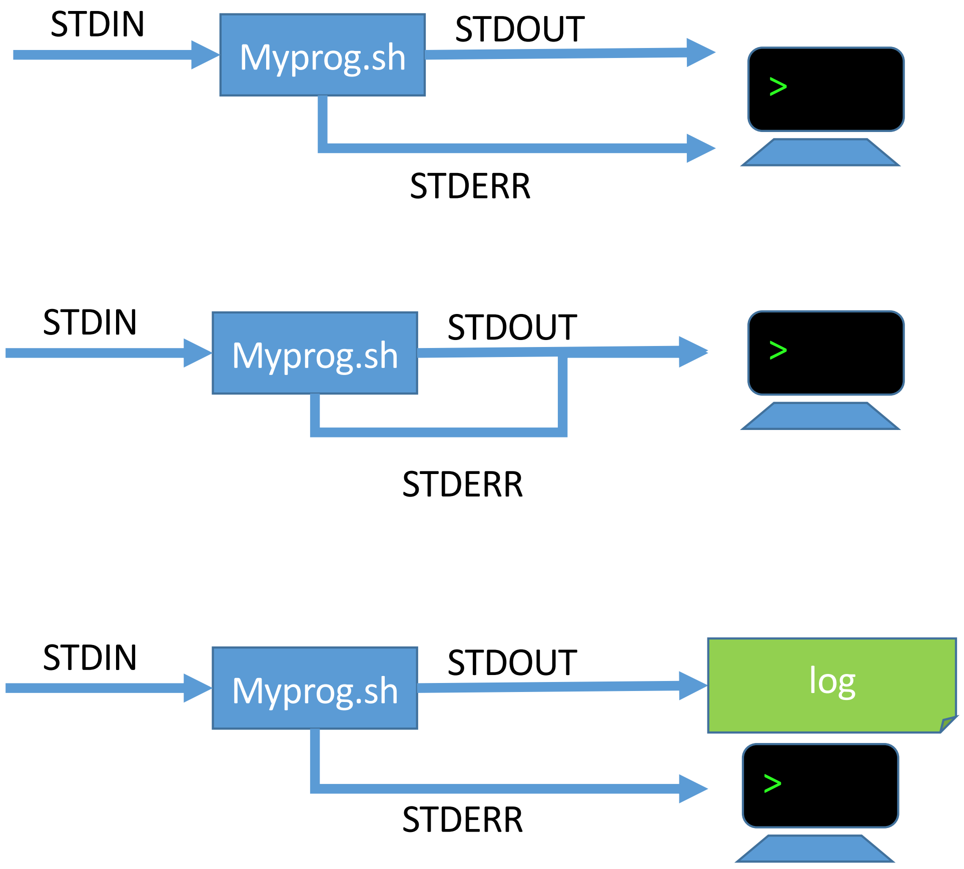
Let's switch it around 1 more time!
(To make sure that we understand)

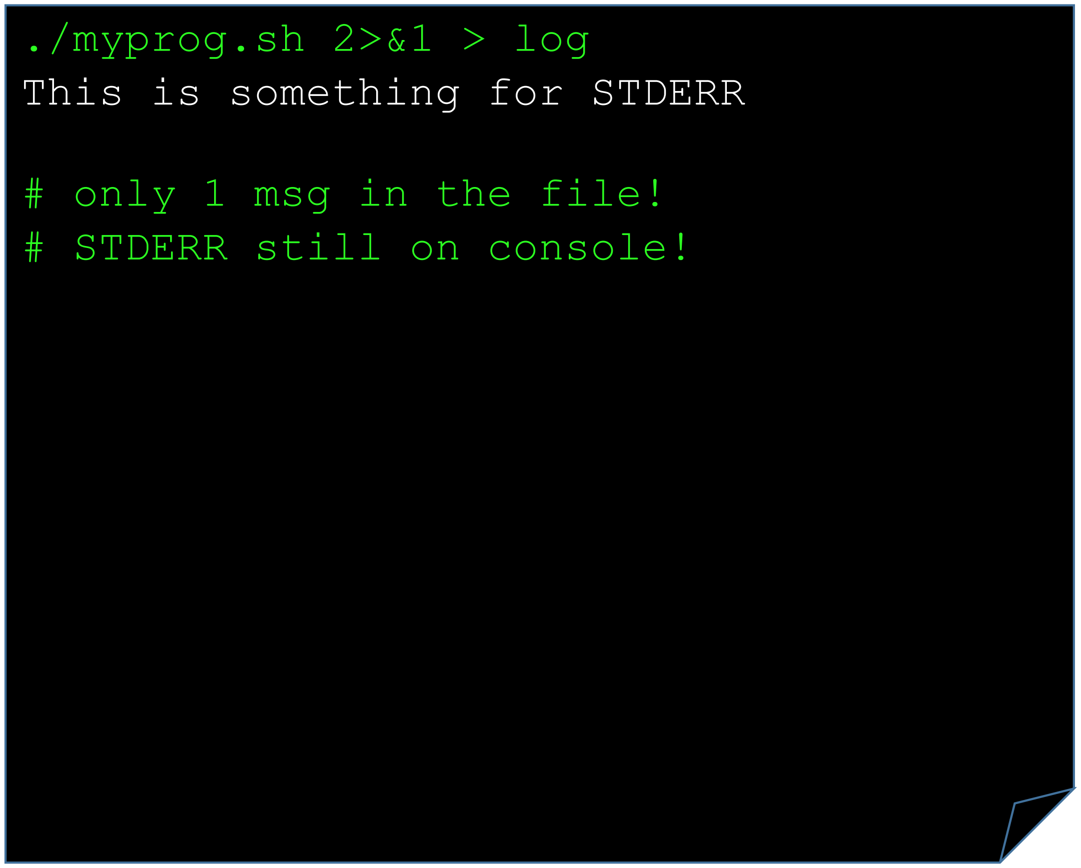




```
./myprog.sh 2>/dev/null 2>&1
This is something for STDOUT
This is something for STDERR
```

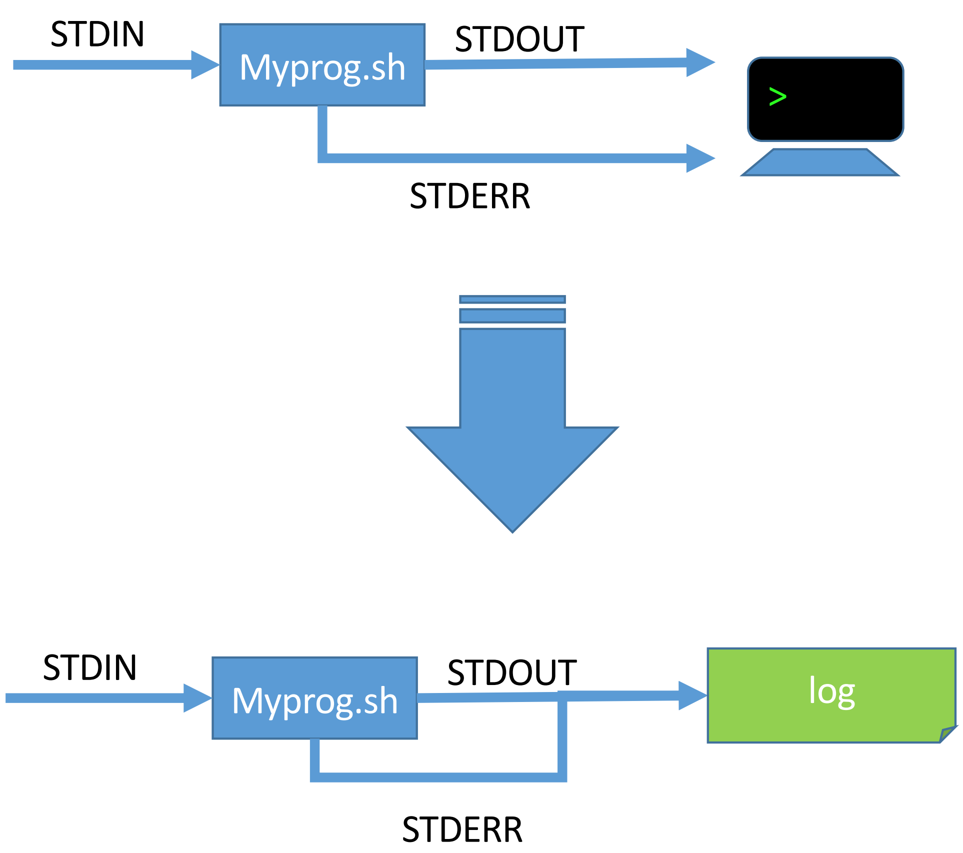
What if we 2>&1 but THEN change 1> instead…





```
./myprog.sh 2>&1 > log
This is something for STDERR

# only 1 msg in the file!
# STDERR still on console!
```

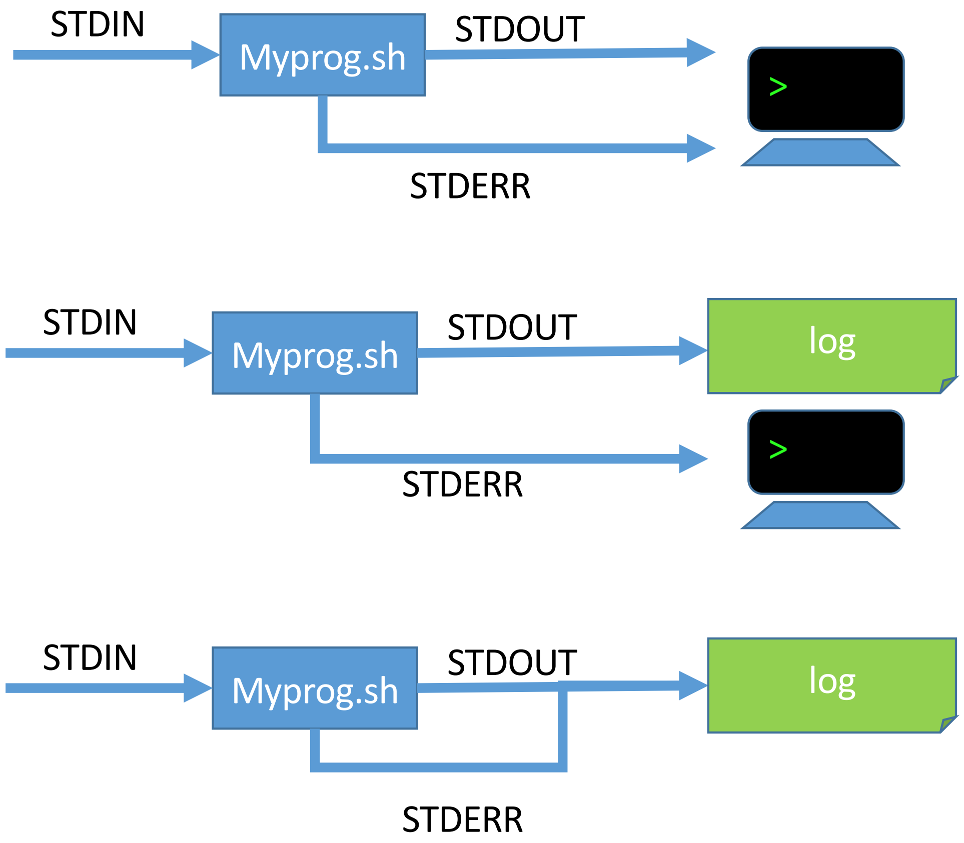
We wanted something different…

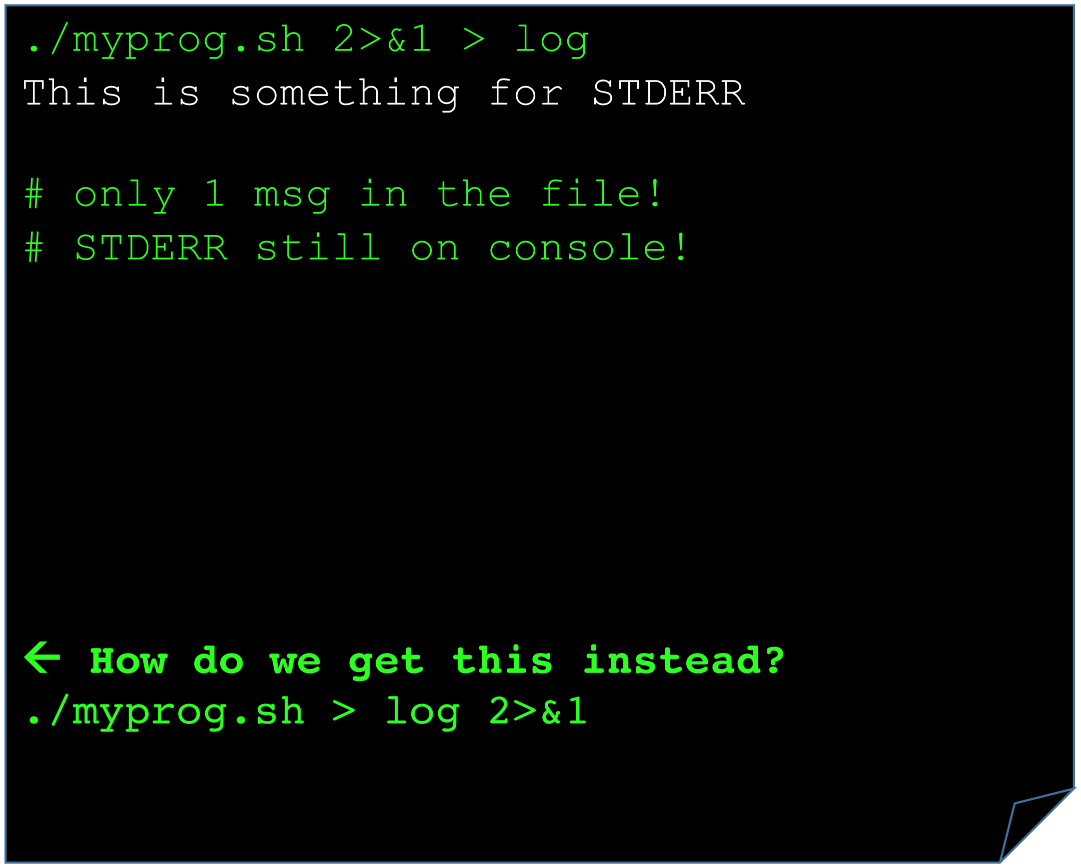


```
./myprog.sh 2>&1 > log
This is something for STDERR

# only 1 msg in the file!
# STDERR still on console!
```

← How do we get this instead?

Here’s how to get it!





```
./myprog.sh 2>&1 > log
This is something for STDERR

# only 1 msg in the file!
# STDERR still on console!
```

← How do we get this instead?

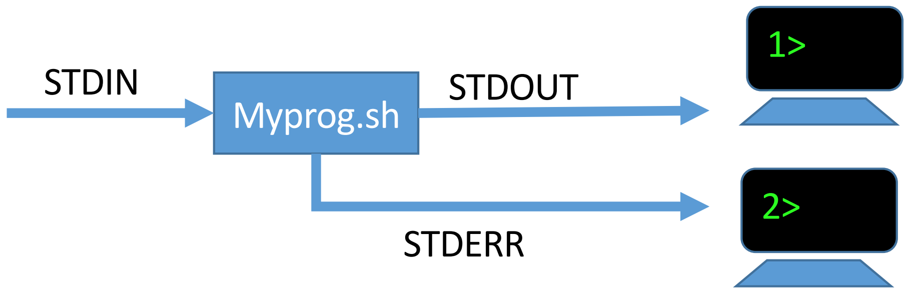
```
./myprog.sh > log 2>&1
```

Topic

M3T1.4 Bash Redirections – Swapping STDOUT & STDERR

https://youtu.be/0kY9nFO1078

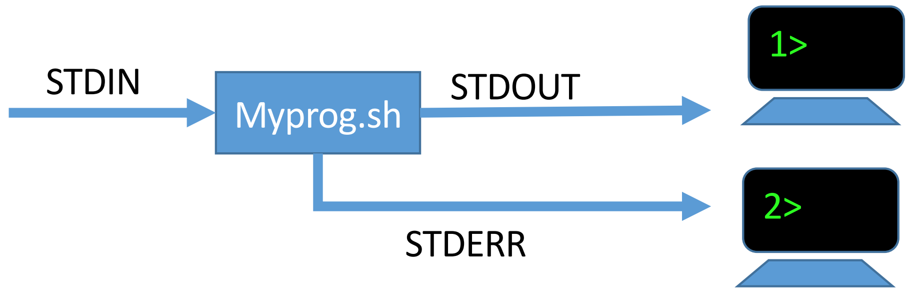
Challenge: how to swap STDOUT / STDERR

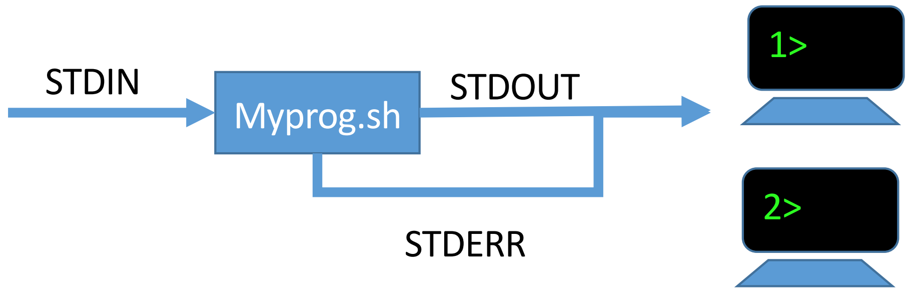


Let’s start by only using what we learned so far

• >	1>	2>
• >>	1>>	2>>
• 1>&2	2>&1	

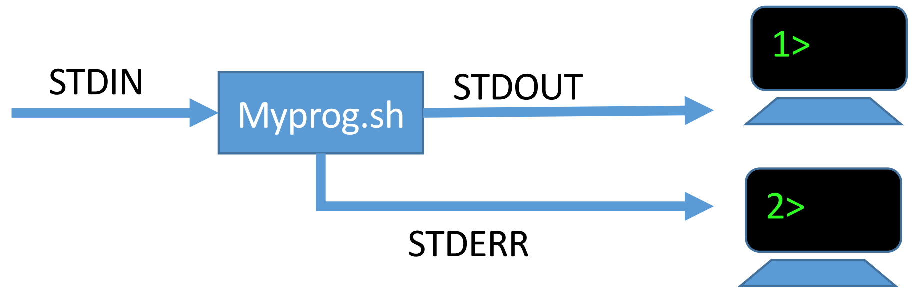
Trying 2>&1

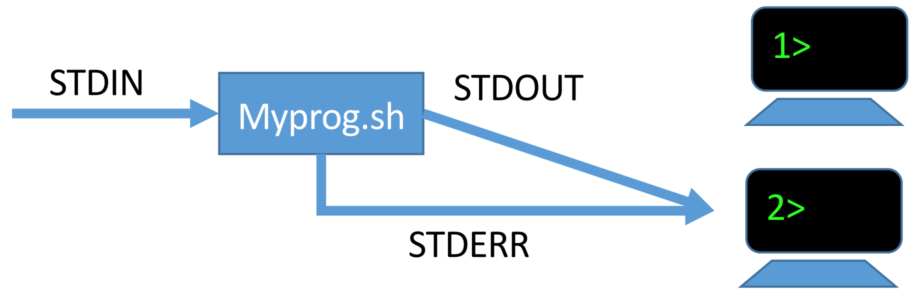




We lost STDERR target, whether it was the console or a file or a pipe…

Trying 1>&2





We lost STDOUT target, whether it was the console or a file or a pipe…

Problem is reminiscent of…

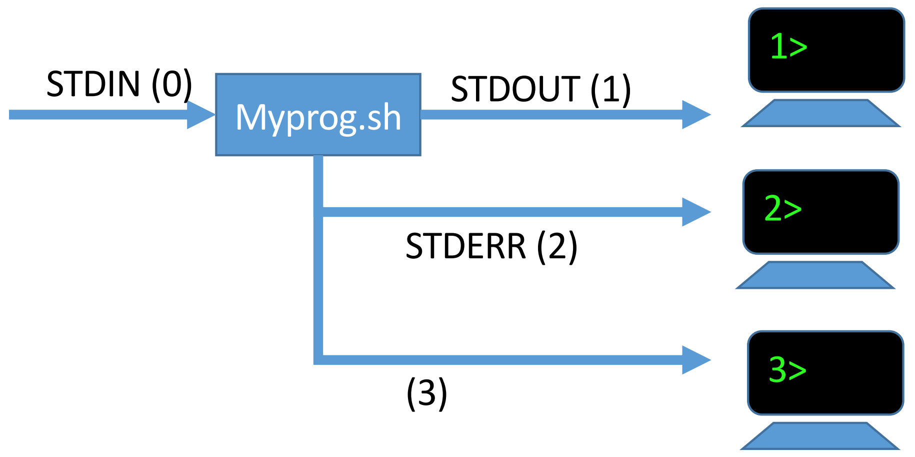
- Given two variables x and y, swap their contents.

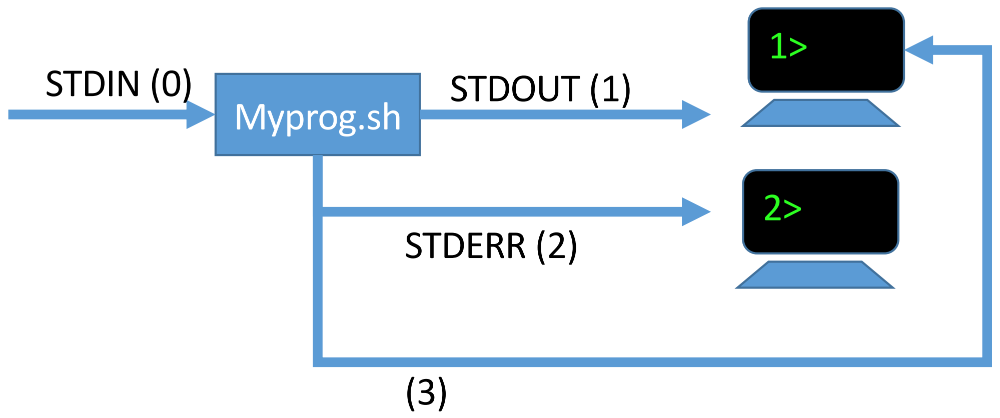
- x = y ; y = x → we lost the value that was in x
- y = x ; x = y → we lost the value that was in y

- tmp = y ; y = x ; x = tmp → this works

Solution (1/3)

- 0, 1, and 2 are indexes in the file descriptors table
- There is a file descriptor at index 3 that we could use as TMP

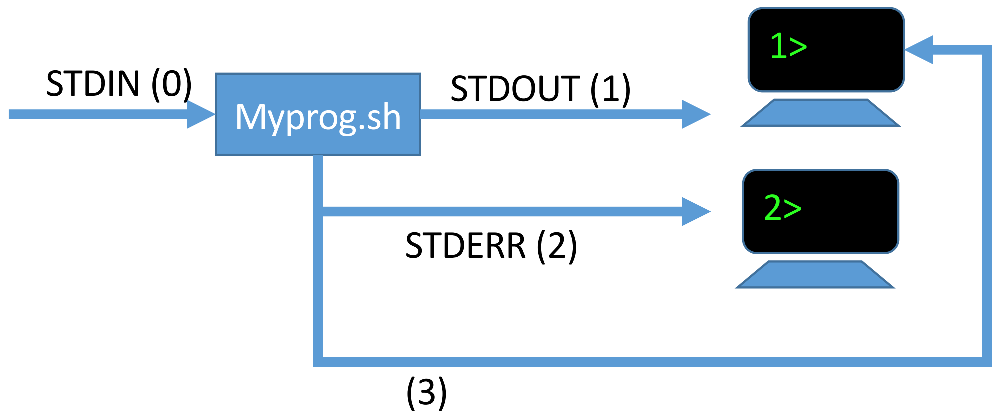


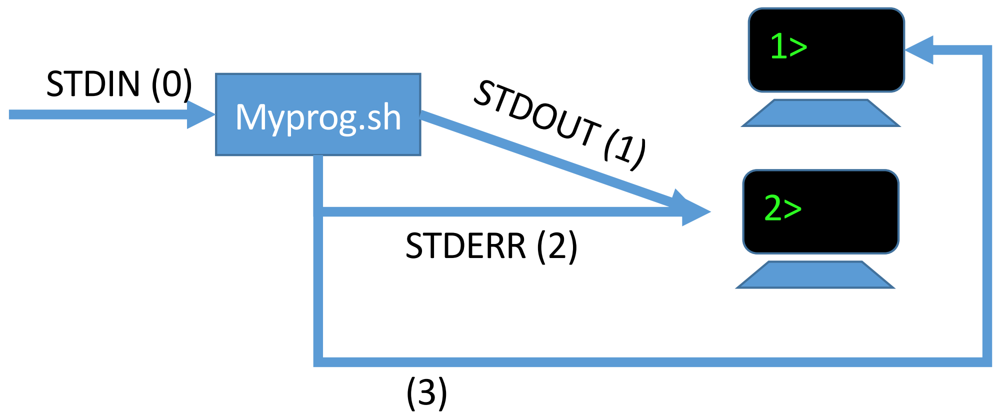


- 3>&1 → saves FD #1 in FD #3

Solution (2/3)

- 0, 1, and 2 are indexes in the file descriptors table
- There is a file descriptor at index 3 that we could use as TMP

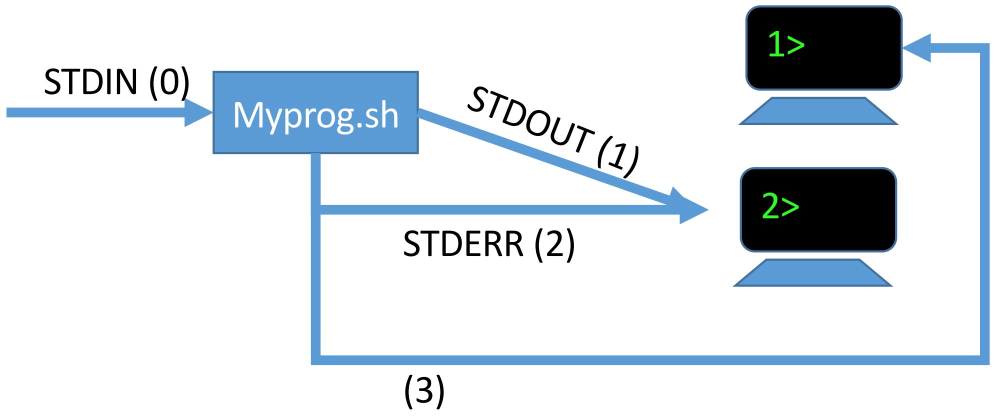


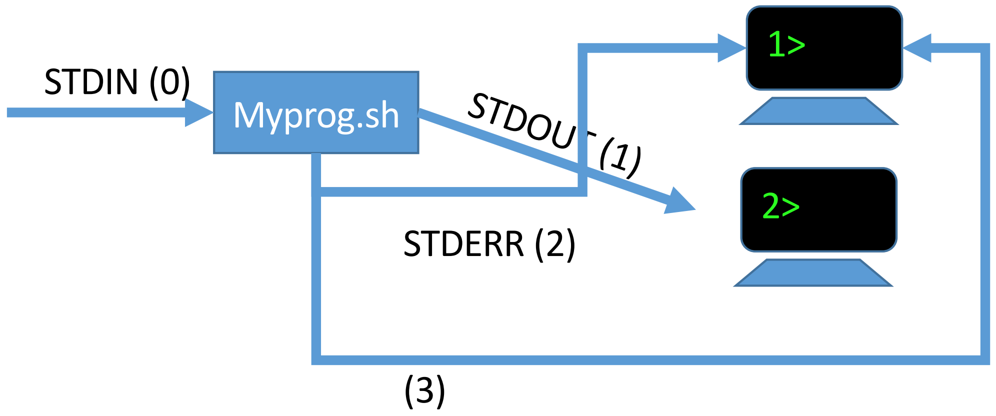


- 3>&1   1>&2

Solution. (3/3)

- 0, 1, and 2 are indexes in the file descriptors table
- There is a file descriptor at index 3 that we could use as TMP





- 3>&1 1>&2 2>&3

Let’s apply this to myprog.sh

- Hardest thing we can do with what we learned so far so good to wrap up the topic
- Practical application: swapping two FDs
- Generalizes what we learned so far: 0,1,2 are not the only FDs available!

How do we verify that the swap really happened?

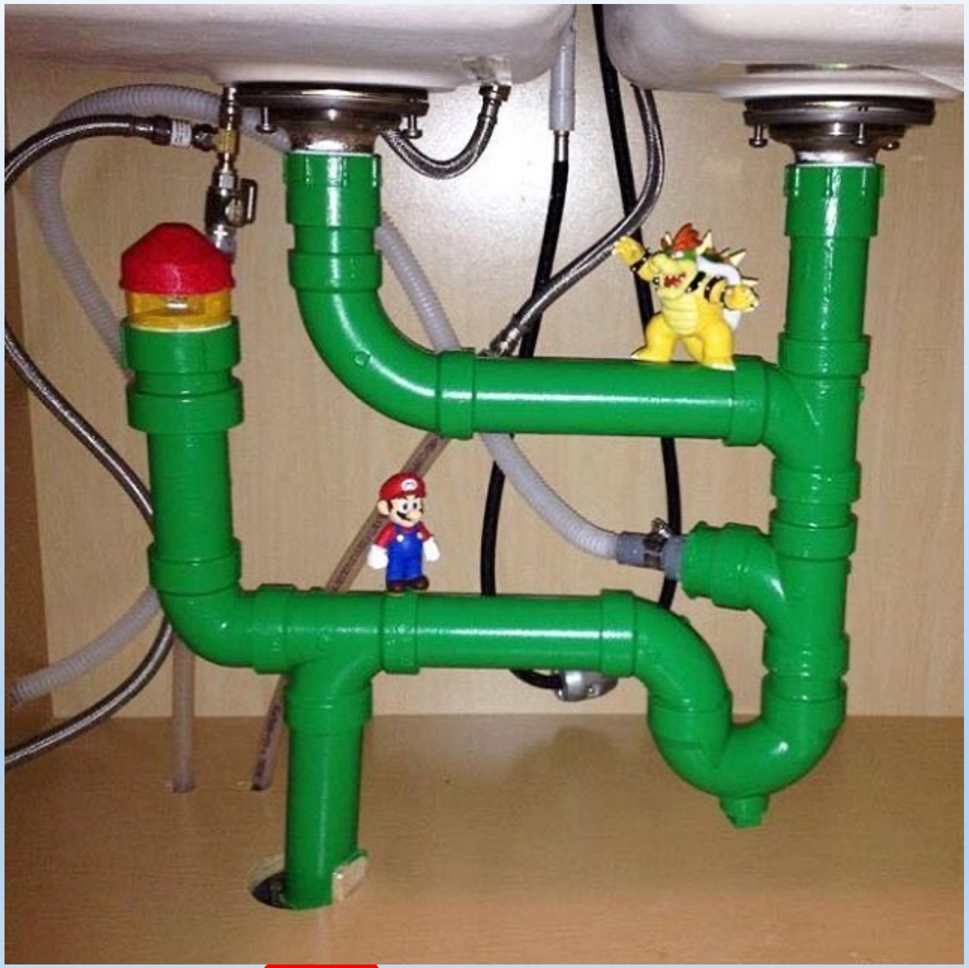
```
./myprog.sh 3>&1 1>&2 2>&3
This is something for STDOUT
This is something for STDERR

# Cannot really tell that it worked
# so we try something more…
```

Let’s verify that the swap really happened

- Hardest thing we can do with what we learned so far so good to wrap up the topic
- Practical application: swapping two FDs
- Generalizes what we learned so far: 0,1,2 are not the only FDs available!

```
./myprog.sh 3>&1 1>&2 2>&3
This is something for STDOUT
This is something for STDERR

# Cannot really tell that it worked
# so we try something more…

./myprog.sh > out 2> err 3>&1 1>&2 2>&3
cat out
This is something for STDERR
cat err
This is something for STDOUT
```

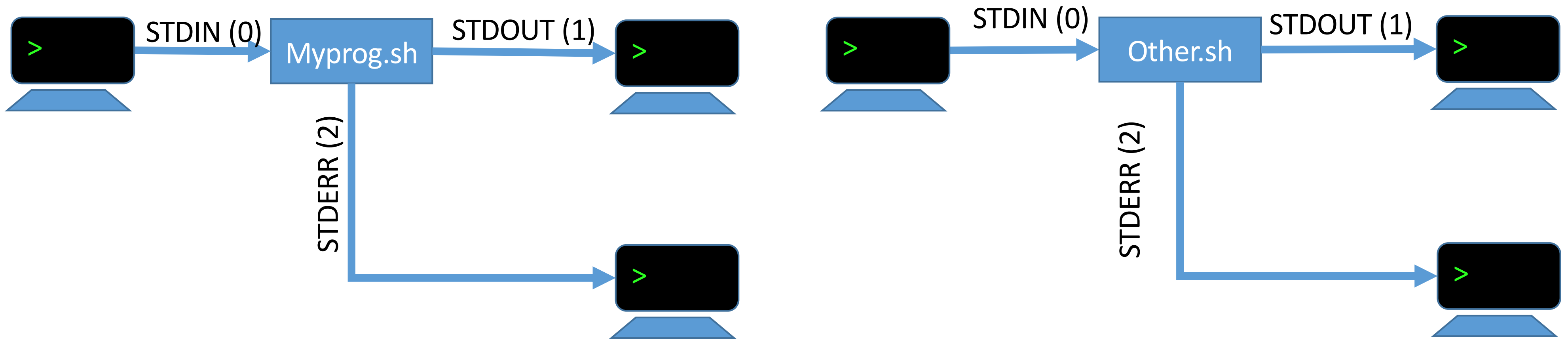
M3T1.5 Bash Redirections – Piping



https://youtu.be/pPEX5q6odFI

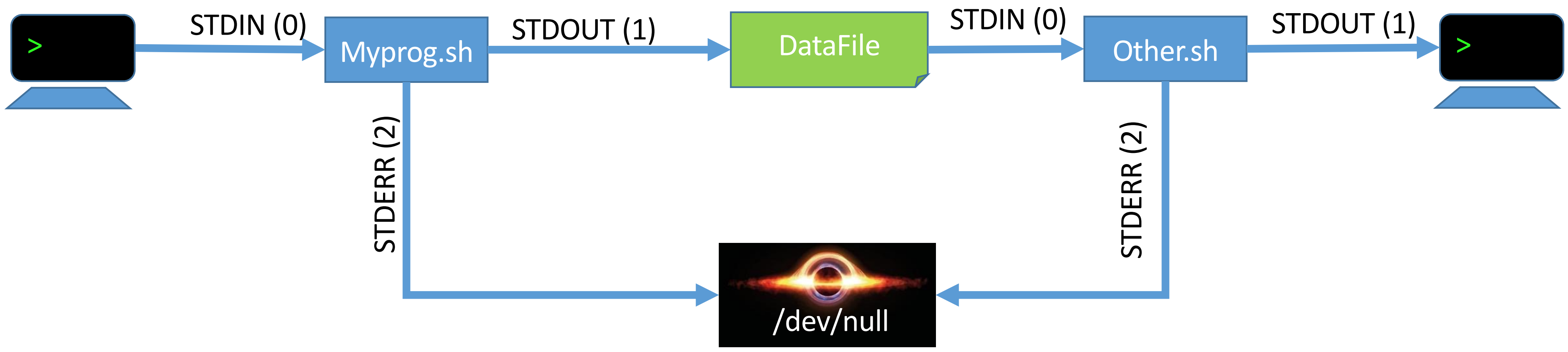
The story so far…

- Connecting FILES to STDOUT or STDERR
- What if we have two processes…



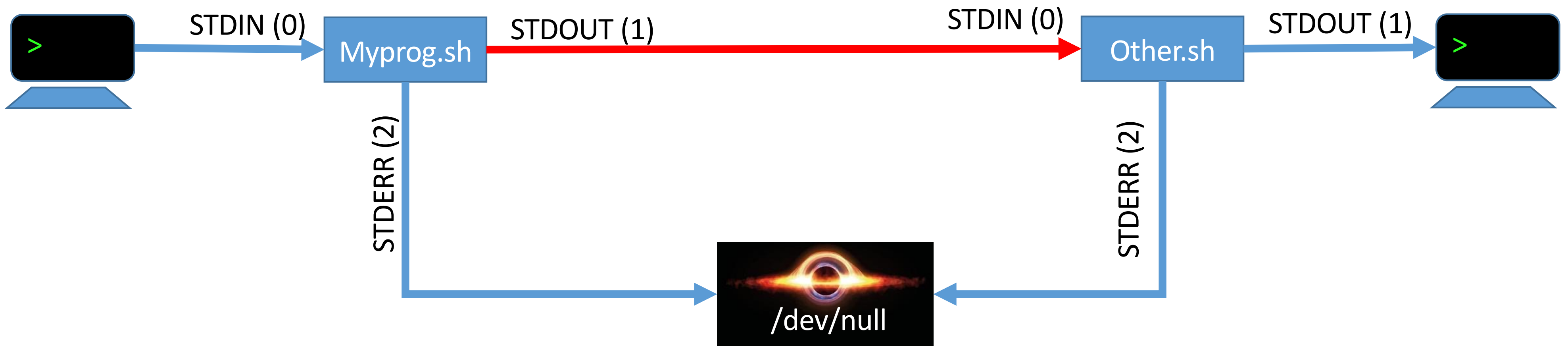
The story so far…

- Connecting FILES to STDOUT or STDERR
- What if we have two processes…
- Let's use DataFile to send data from 1st process to 2nd process



What about we skip using DataFile?

- Using | instead
- How do we illustrate this with simple programs?



Let’s find some commands to illustrate this

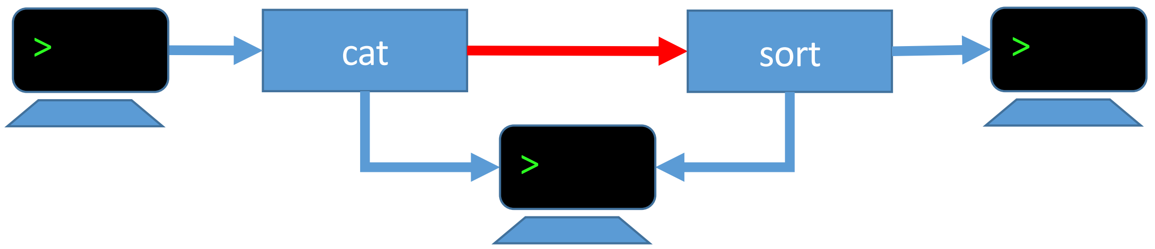
```
cat
One
One
Two
Two
^D

cat > somedata.txt
One
Two
Three
^D
cat somedata.txt
One
Two
Three
```

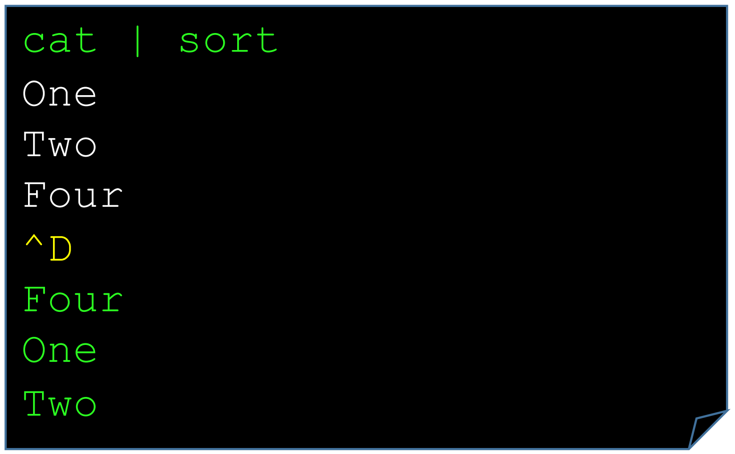
We need a command that reads from STDIN (kbd) and displays on STDOUT (screen)

We also need a command that reads its input from STDIN and displays its output to STDOUT

```
sort
One
Two
Four
^D
Four
One
Two
```

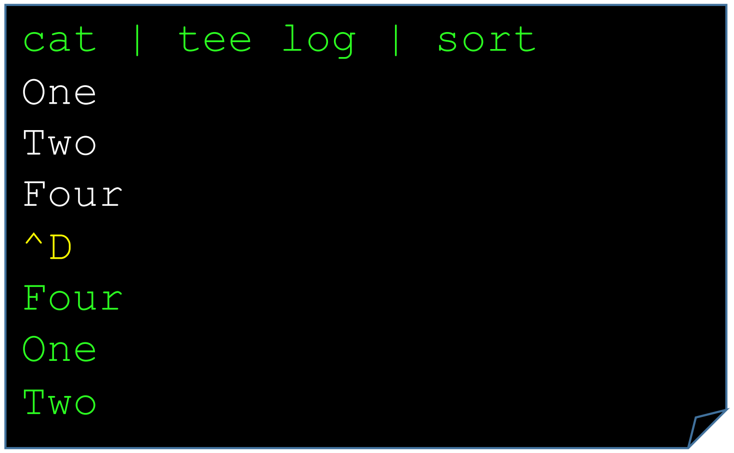
cat & sort example:



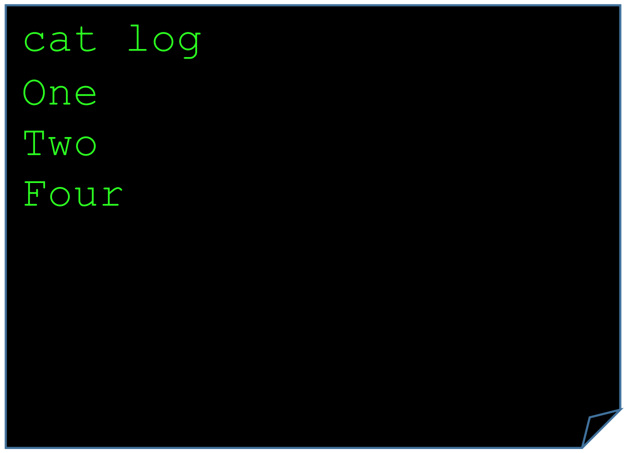


```
cat | sort
One
Two
Four
^D
Four
One
Two
```

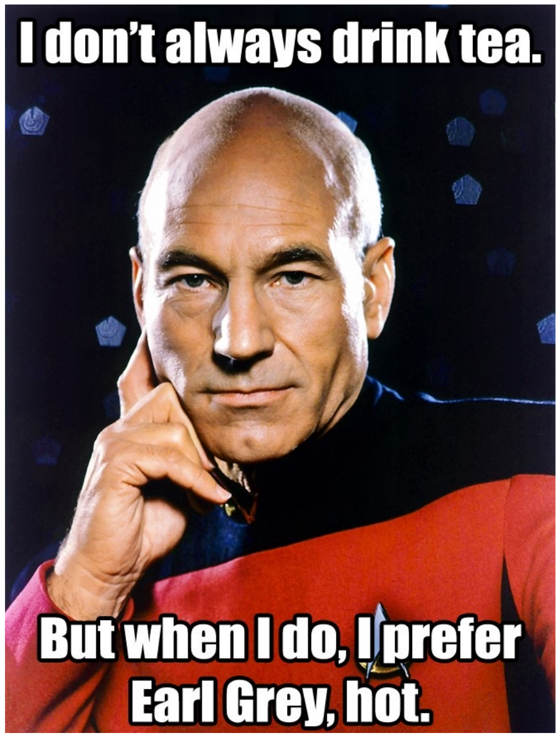
Introducing Tee. Earl Grey. Hot.

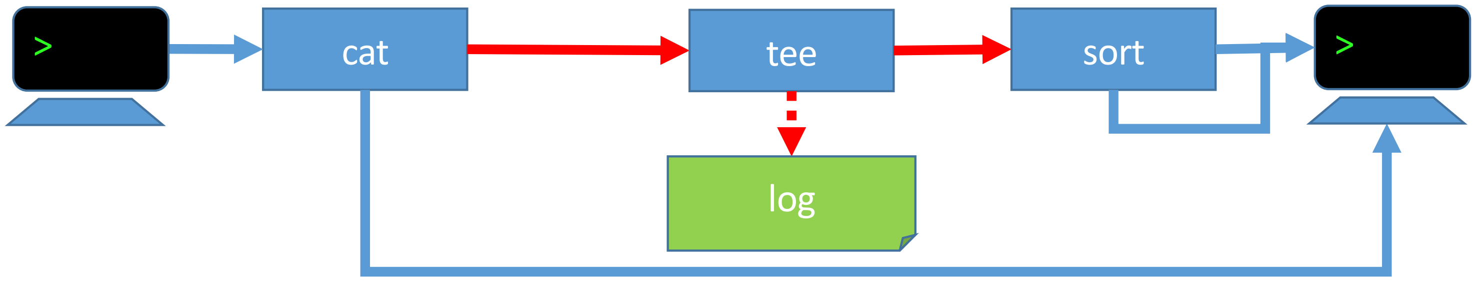


```
cat | tee log | sort
One
Two
Four
^D
Four
One
Two
```

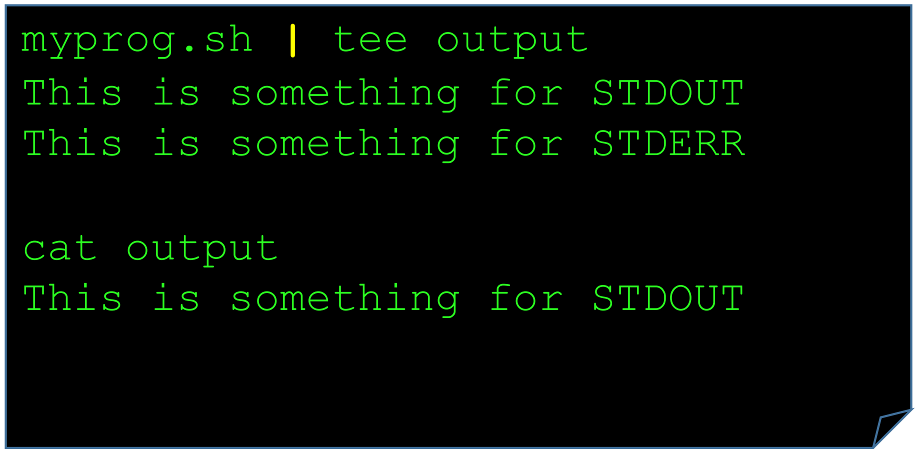


```
cat log
One
Two
Four
```

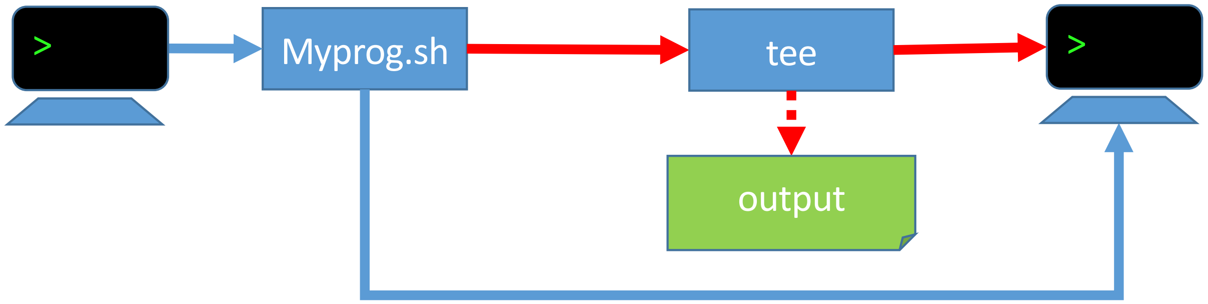

I don't always drink tea.
But when I do, I prefer Earl Grey, hot.



How to pipe both STDOUT and STDERR?



```
myprog.sh | tee output
This is something for STDOUT
This is something for STDERR

cat output
This is something for STDOUT
```

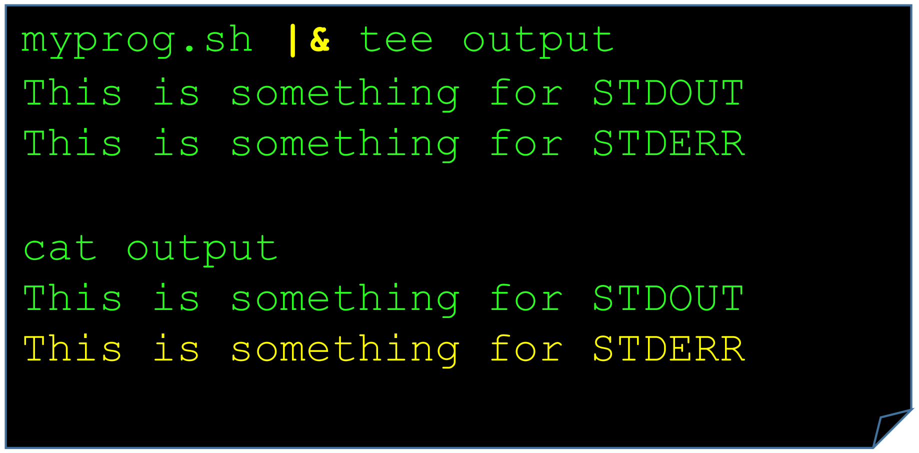


- This is what happens with just a regular |

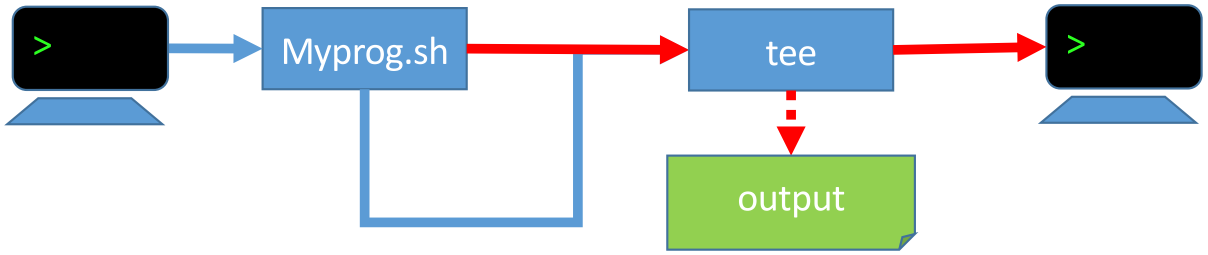
How to pipe both STDOUT and STDERR?

Not working on MacOS

- Now, we use the |& operator instead
- Both STDOUT and STDERR of myprog.sh were redirected to the STDIN of tee
- The STDOUT of tee, as well as the file, contain both messages



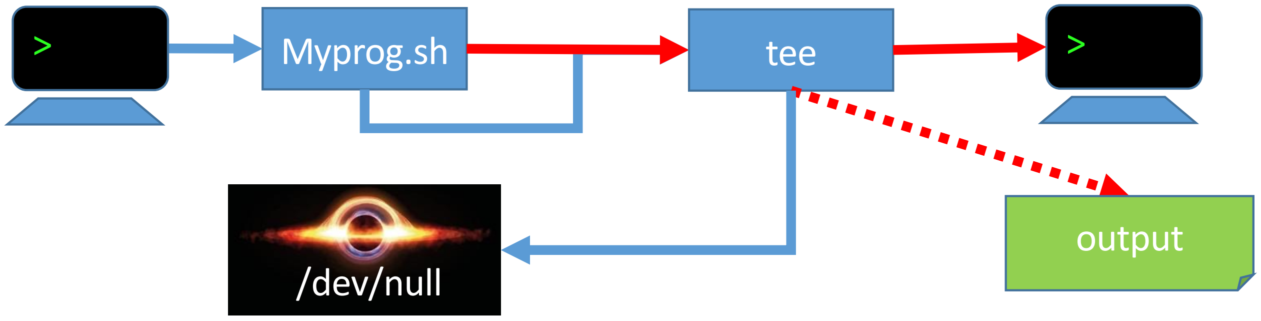
```
myprog.sh |& tee output
This is something for STDOUT
This is something for STDERR

cat output
This is something for STDOUT
This is something for STDERR
```



How do we make sure both STDOUT & STDERR ended up in STDOUT of tee?

```
myprog.sh |& tee output 2> /dev/null
This is something for STDOUT
This is something for STDERR

cat output
This is something for STDOUT
This is something for STDERR
```

- We redirect STDERR of tee to /dev/null
- We still get both msgs on screen
- Therefore, there was nothing on STDOUT coming out of tee



Topic

Interlude: PA2b

These Practice Exercises are meant to help you review for IE2.ß

Counting bashes

#1

- How do I use piping to count the number of bash interpreters running on my machine?
- Hint: we used the commands `wc` and `grep` in previous slide examples…

Each step its own log file

#2

- I want to run a multi-steps pipeline of commands but keep the STDOUT at each step in a file out.1, out.2, out.3, out.4 …
- For example, I want to filter out of a dictionary file all words not containing a letter 'a', then do the same on the result with words not containing the letter 'b', and keep going like this until I have only on STDOUT the words that contain all vowels

```
cat /usr/share/dict/words | grep a | grep e | grep i | grep o
| grep u | grep y
```

- How do I save each intermediary step's STDOUT?

What happens here?

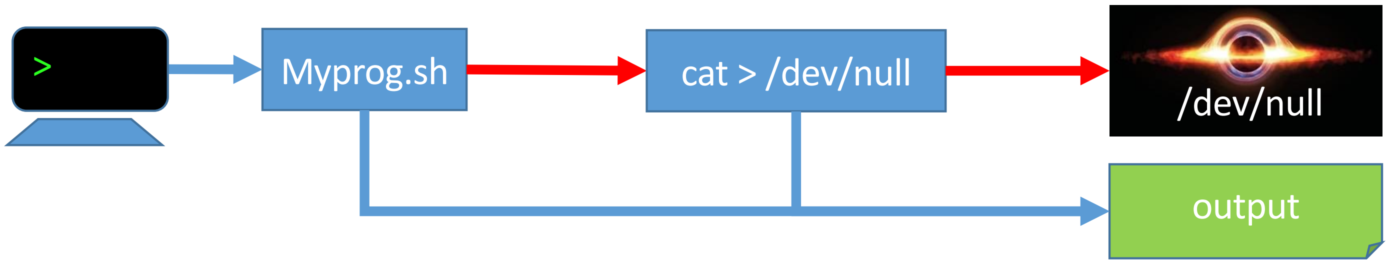
#3

```
./myprog.sh 1>out 3>&1 1>&2 2>/dev/null 1>&3
```

~~Homer's~~ STDERR ~~triple~~ Bypass

#4

This is what we want:


>
Myprog.sh
cat > /dev/null
/dev/null
output