# Quick Announcements


## Update on Case Study

- 1 student per team sends, via canvas message to the TAs, the following information:
  - List of team members
  - Topic chosen by the team
  - Description (1-2 sentences) of each member's part

## "warm up" PAs from previous slide PDF

- All are trivial except the "It's meta" one
- Post on piazza if you have questions about any of them

# It's meta.

- We want to assign to a variable called `META` the value `$RANDOM`. Please note, we don't want to assign a random number but the value `$RANDOM` as a string. How do you do that?
- We then want to display a message that states: Your lucky number is `112233`. The value `112233` will be a random number, different every time we re-execute the line, and generated by using the variable `META`

```
$ echo  $( echo $META)
$RANDOM

$ echo $(echo $( echo $META))
$RANDOM
```

Why doesn't the following work?

How do we make it work?

- Hint: check out the doc for `eval`

Module

# M03
# The UNIX Way

# Menu for this module

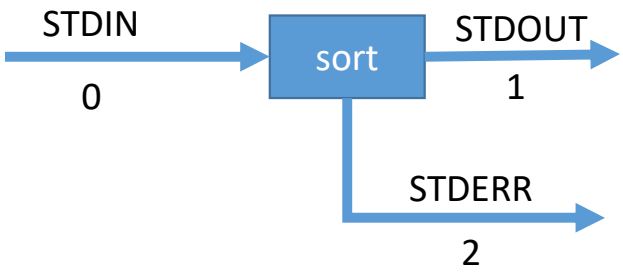
| | | |
|---|---|---|
| T1 | Redirections & Piping | It is possible, and useful, to direct the text output of a given command line tool so that it is accepted as input of another. There "redirections" are part of what makes command line useful to automate tasks. |
| T2 | Filters | For the above to be really leveraged, we need tools which are designed to work line per line on the data fed to them from the keyboard, a file, or another tool’s output. Enter the filters.... |

Topic

# M3T1.1
# Bash Redirections Rudiments

https://youtu.be/yV0wCSm2YBY

# Bash Redirections - Rudiments

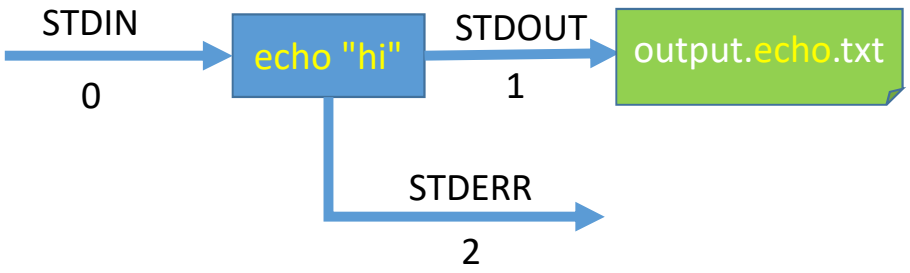



- Pipelines of information between processes
- Each process has 3 file descriptors

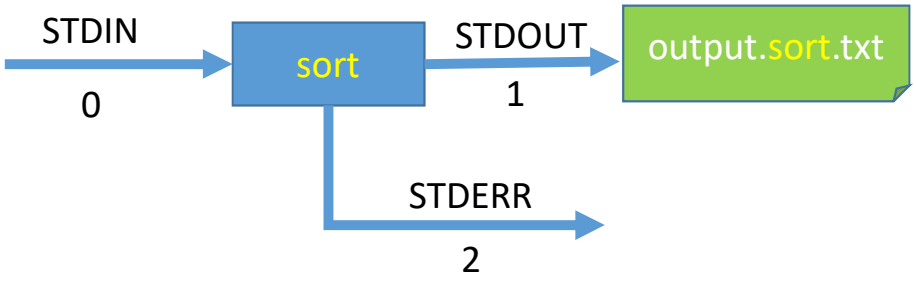
```
$ sort
one
two
three
^D
one
three
two
```

```
$ sort -l
sort: invalid option -- 'l'
Try 'sort --help' for more
information.
```

# Redirecting STDOUT to file



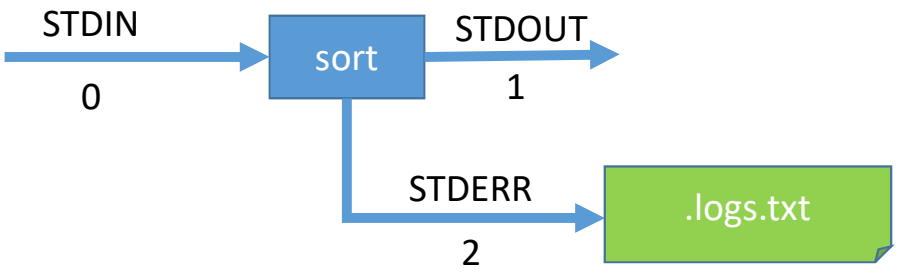

```
$ echo "hi" > output.echo.txt
```




```
$ sort 1> output.sort.txt
One
Two
Three
^D
```

NOTE – STDERR is still displaying to the console with a `sort -l`

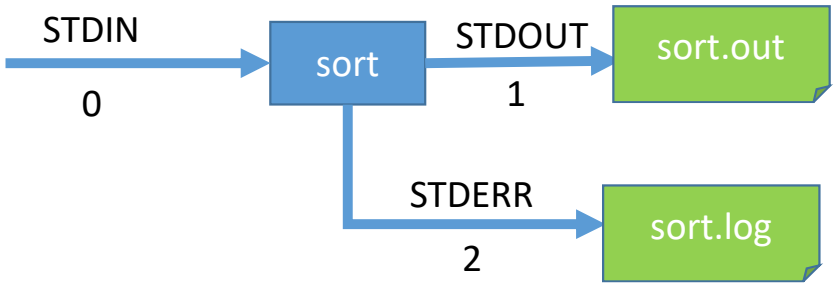
# Redirecting STDERR to file




```
$ sort -l > logs.txt
$ cat logs.txt

$ sort -l 1> logs.txt
$ cat logs.txt

$ sort -l 2> logs.txt
$ cat logs.txt
```

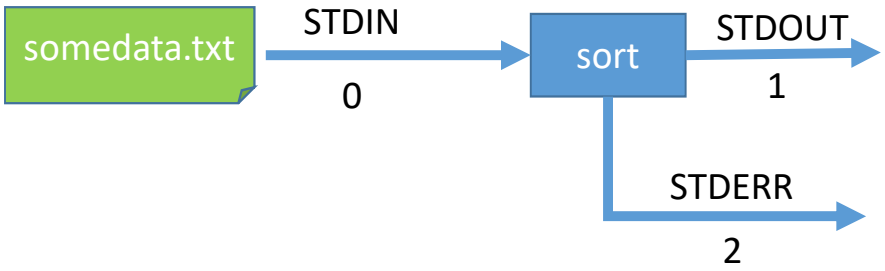
- > and 1> do not work for this
- We need a new syntax: 2>

# Redirecting both STDOUT & STDERR to 2 files




```
$ sort -l 2> sort.log 1> sort.out
$ cat sort.log
$ cat sort.out

$ # does order matter?

$ echo "hi" 2> echo.log 1> echo.out
$ cat echo.log
$ cat echo.out

$ echo "hi" 1> echo.out 2> echo.log
$ cat echo.log
$ cat echo.out

$ # order does not matter
```

# What about STDIN?




```
$ cat > somedata.txt
We are
Making a new file
With cat redirecting its output
^D

$ sort < somedata.txt
$ sort 0< somedata.txt

$ sort -l < somedata.txt
sort: invalid option -- 'l'
Try 'sort --help' for more information.
```

# Let's "Fire Everything!!!" that we got so far

```
$ sort < somedata.txt 2> sort.log 1> sort.out

$ cat sort.log
$ cat sort.out
```

# /dev/null

```
$ cat /dev/null

$ sort -l 2> /dev/null

$ echo "into the black hole!" 1> /dev/null

$ echo "into the black hole!" > /dev/null
```

Topic

# M3T1.2
# Bash Redirections - Appending

https://youtu.be/NknAcynU4sM

# Accumulating STDERR msgs to log file

| | | | |
|---|---|---|---|
| > | 1> | 2> | means overwrite |
| >> | 1>> | 2>> | means append |

```
sort -l 2> sort.err
sort -l 2> sort.err
cat sort.err

# only 2nd message is there

sort -l 2>> sort.err
sort -l 2>> sort.err
cat sort.err

# both messages are there
```

# Trying now with STDOUT

- Works the same way

```
echo "hello" > echo.out
cat echo.out

echo "hello" 1> echo.out
echo "world" 1> echo.out
cat echo.out

# only 2nd message is there

rm echo.out

echo "hello" >> echo.out
echo "world" >> echo.out
cat echo.out

# both messages are there
```

# Bash option to control that behavior

- If we redirect with overwrite to an existing file, I want to prevent losing all existing data

```
set -o noclobber

echo "hello world" > echo.out
Bash: echo.out: cannot overwrite
existing file

cat echo.out #all is still there


set +o noclobber

echo "hello world" > echo.out

cat echo.out #data is gone!!!!
```

# Overriding the safety option

- If we redirect with overwrite to an existing file, I want to prevent losing all existing data
- What if I want an exception for just one command?
- Works with STDOUT, try it with STDERR it works as well

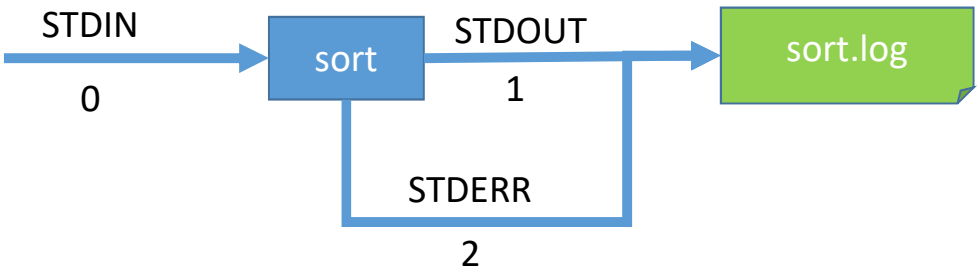
```
set -o noclobber

echo "hello world" > echo.out
Bash: echo.out: cannot overwrite existing file
cat echo.out #all is still there


echo "replacing data by this" >| echo.out
# override noclobber option
cat echo.out
```

Topic

# M3T1.3
# Bash Redirections
# ~~Crossing~~ Merging the Streams

https://youtu.be/92-YIUXGh68

# Sometimes we want STDOUT & STDERR to be redirected to the same location

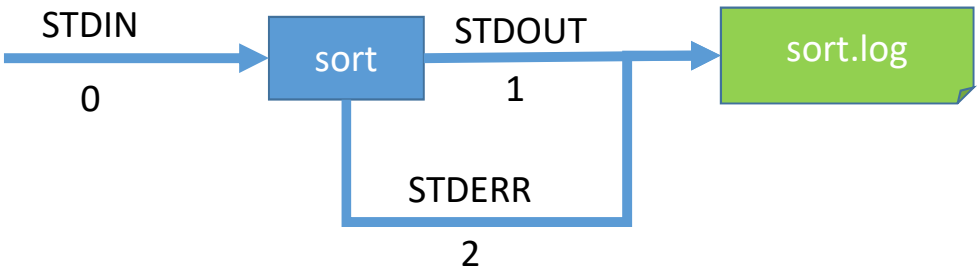



```
sort 1> sort.log 2> sort.log
Bash: sort.log: cannot overwrite
existing file
```

What happened?

Why didn't it work?

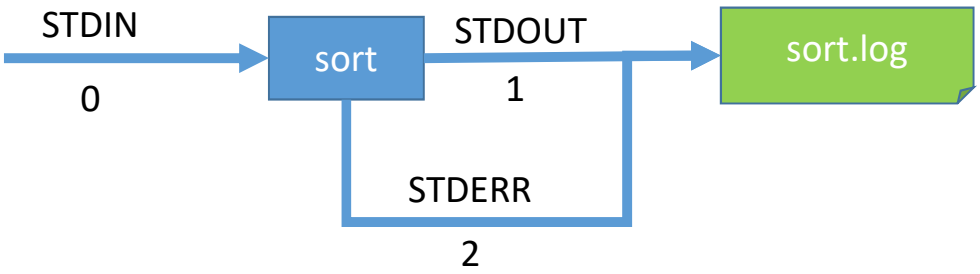
# Here is the explanation…




```
sort 1> sort.log 2> sort.log
Bash: sort.log: cannot overwrite
existing file

# 1st redir created log file
# 2nd redir targeted the same file
```

How do we fix it?

# This is how we fix it…




```
sort 1> sort.log 2> sort.log
Bash: sort.log: cannot overwrite
existing file

# 1st redir created log file
# 2nd redir targeted the same file
# if we get rid of noclobber…
set +o noclobber

sort 1> sort.log 2> sort.log
cat sort.log
# YAY! the output is in the log

sort -l 1> sort.log 2> sort.log
cat sort.log
# YAY! error message is in the log
```

It works when we use STDOUT or STDERR… what about AND?

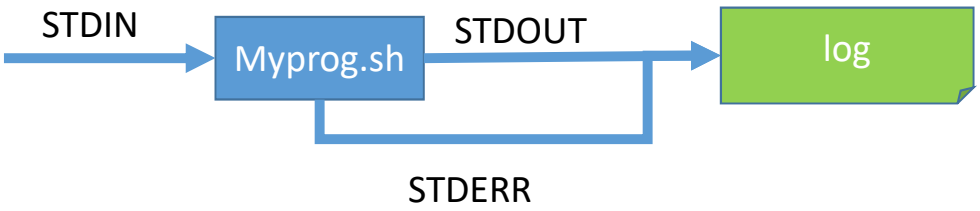
# What if our command generates both STDOUT and STDERR?

- Testing with myprog.sh

```
#!/bin/bash

>&2 echo "This is for STDERR"
echo "This is for STDOUT"
```

```
./myprog.sh
This is something for STDOUT
This is something for STDERR


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

./myprog.sh 1>/dev/null
This is something for STDERR

./myprog.sh 1>/dev/null 2>/dev/null

# → able to redirect BOTH to /dev/null
# But
# what if we redirect to something else?
```

# Let's redirect to a file!
# Problem: 1 of the messages is lost!




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

# only 1 of the messages made it to log
```

## Why did we lose 1 of the 2 messages?

- We turned off noclobber
- So now we have clobbering ☺

# Solution: new syntax

- New Syntax!!!

2>&1

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

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

# Solution: new syntax

- New Syntax!!!

2>&1

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

```
./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?
```

# Solution: new syntax

- 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
```

# Solution: new syntax

- New Syntax!!!

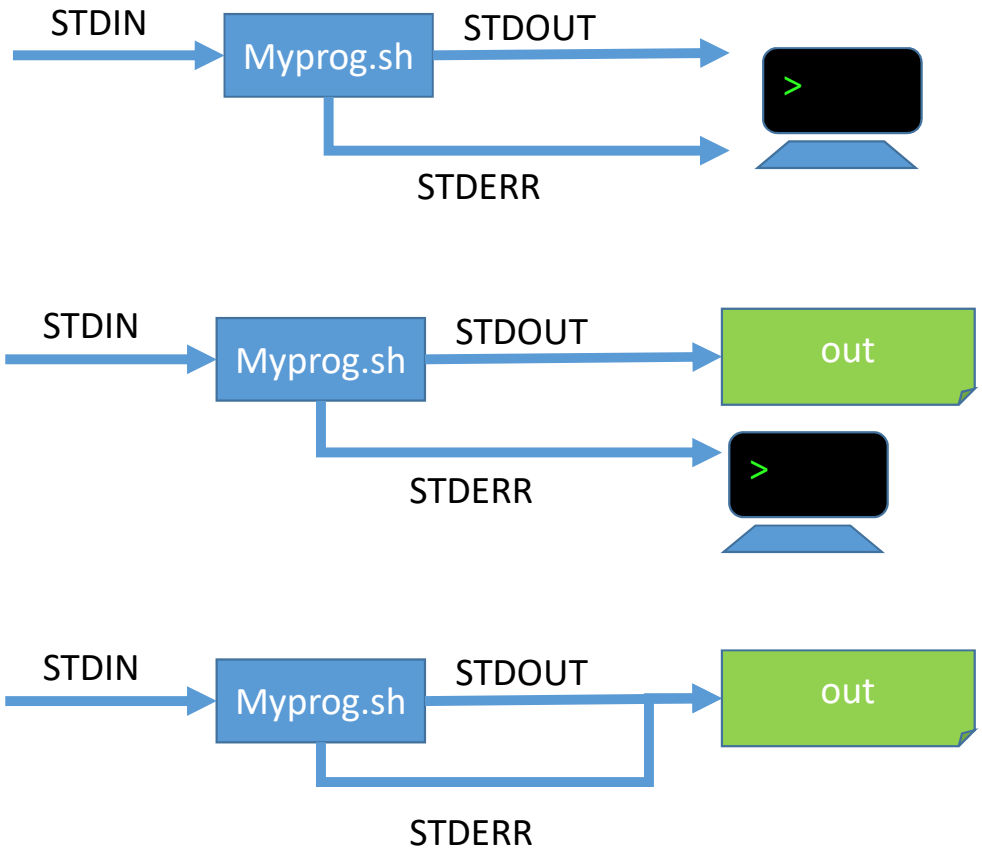
2>&1

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

- New bug!!!! ;p

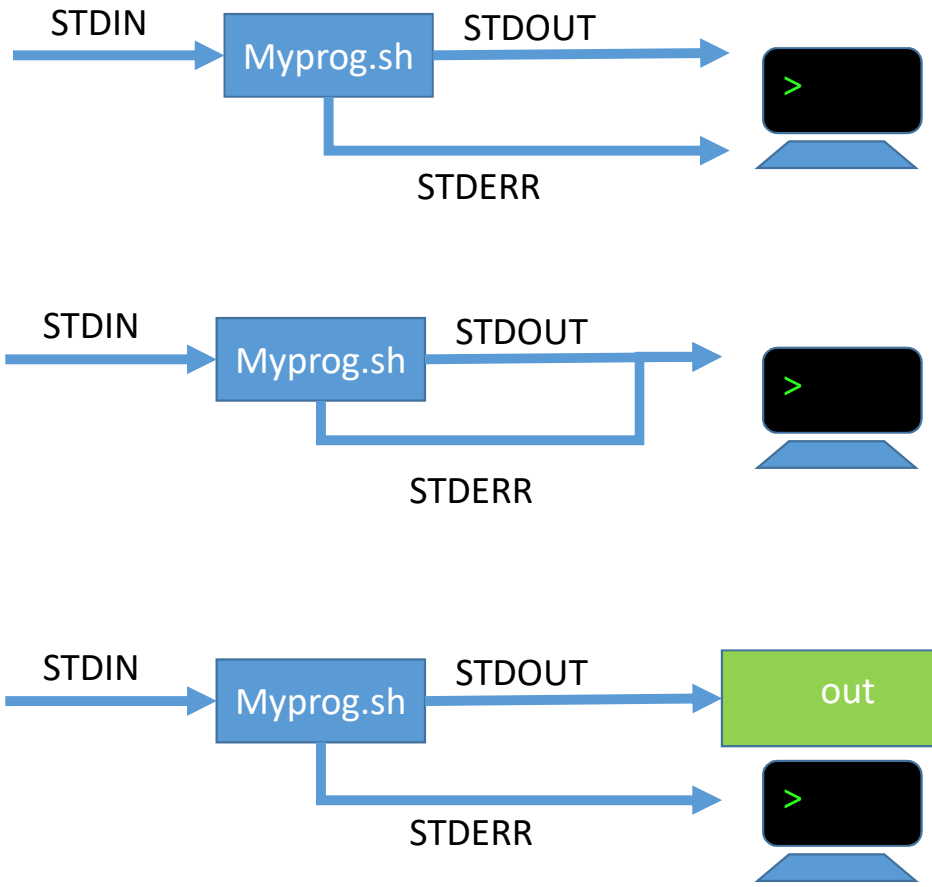
```
./myprog.sh 2>&1
This is something for STDOUT
This is something for STDERR

# 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
```

# Interlude (1/2) – What happens if we re-redirect?



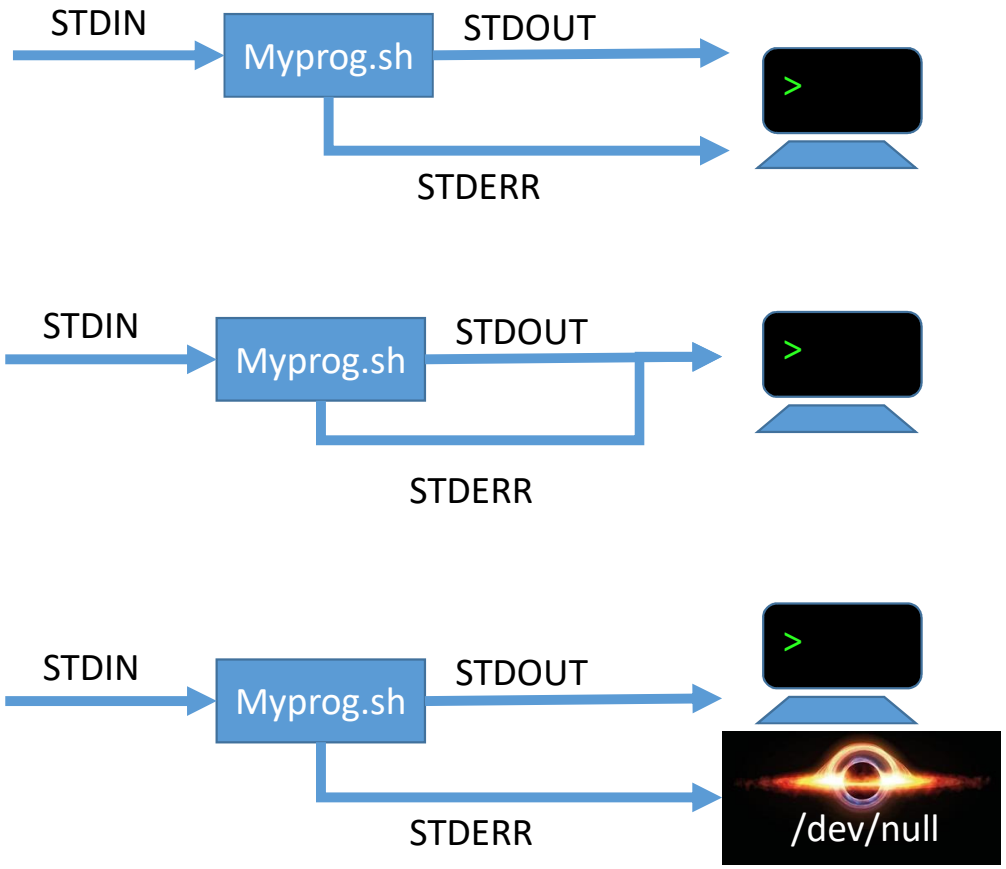

```
./myprog.sh
This is something for STDOUT
This is something for STDERR



./myprog.sh > out
This is something for STDERR
cat out
This is something for STDOUT


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

# Interlude (2/2) – What if we reverse the order?



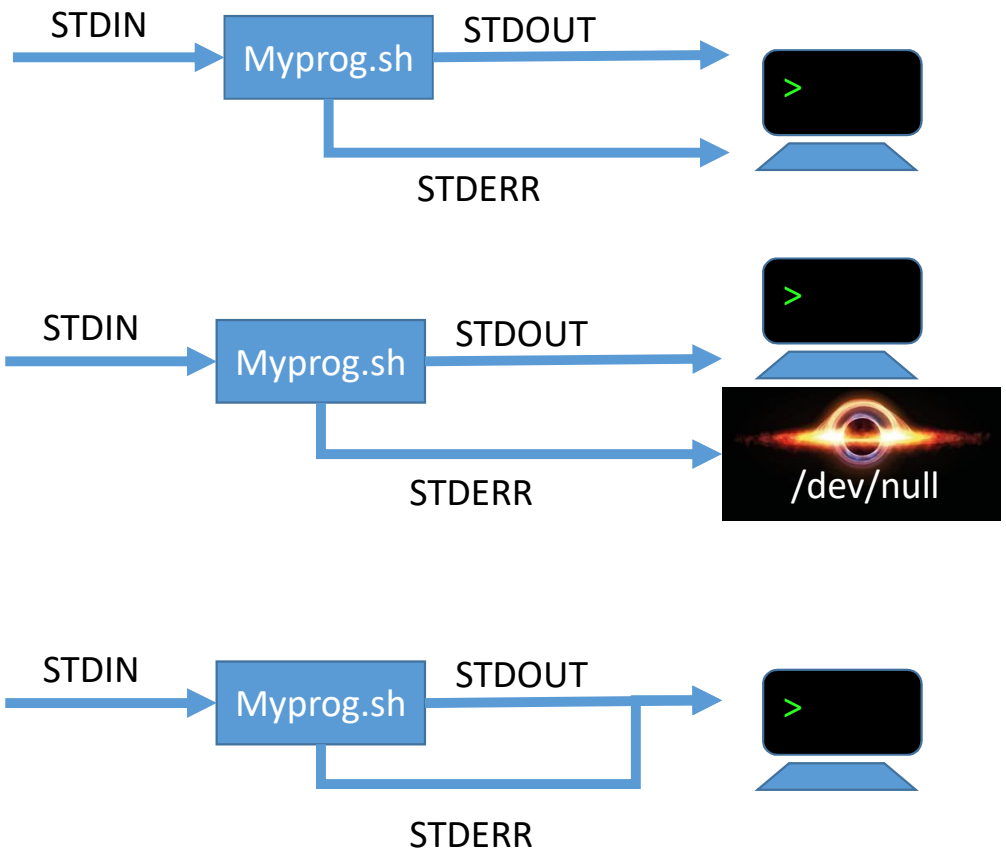

```
./myprog.sh
This is something for STDOUT
This is something for STDERR


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


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

# Back to our redirection to /dev/null



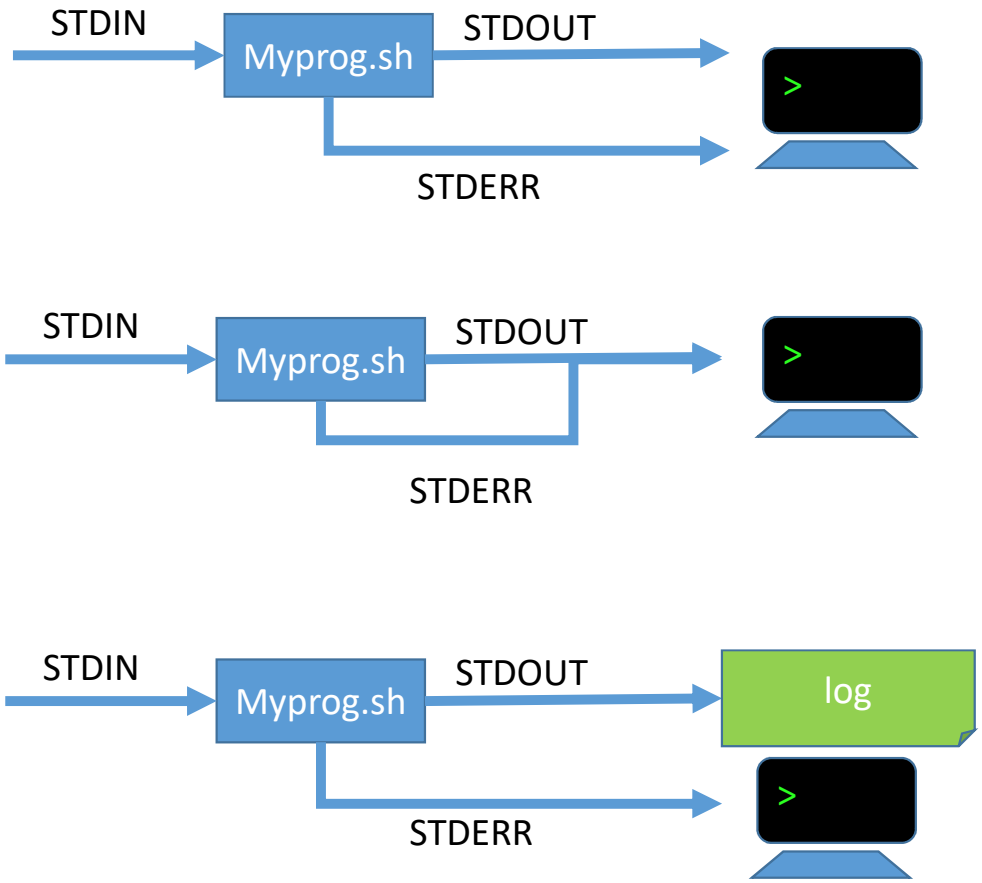

```
./myprog.sh 2>&1 2>/dev/null
This is something for STDOUT
```

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



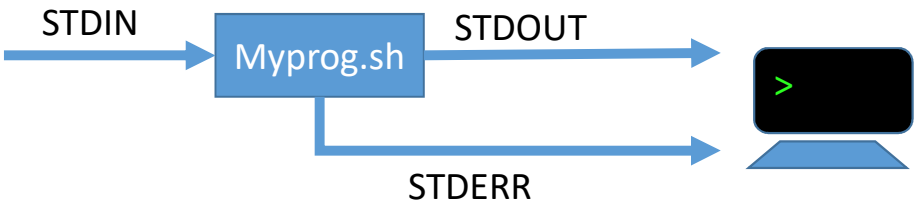

```
./myprog.sh 2>/dev/null 2>&1
This is something for STDOUT
This is something for STDERR
```

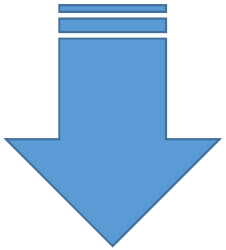
# What if we 2>&1 but THEN change 1>…

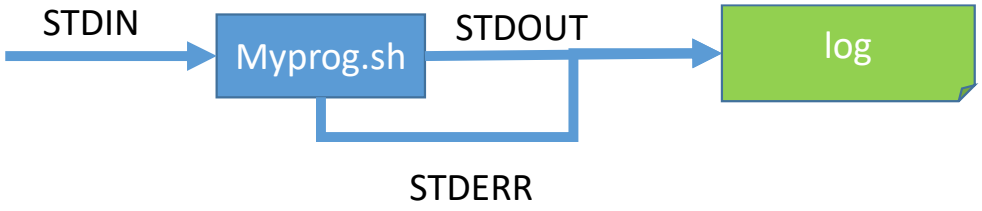



```
./myprog.sh 2>&1 > log
This is something for STDERR

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

# What if we 2>&1 but THEN change 1>…

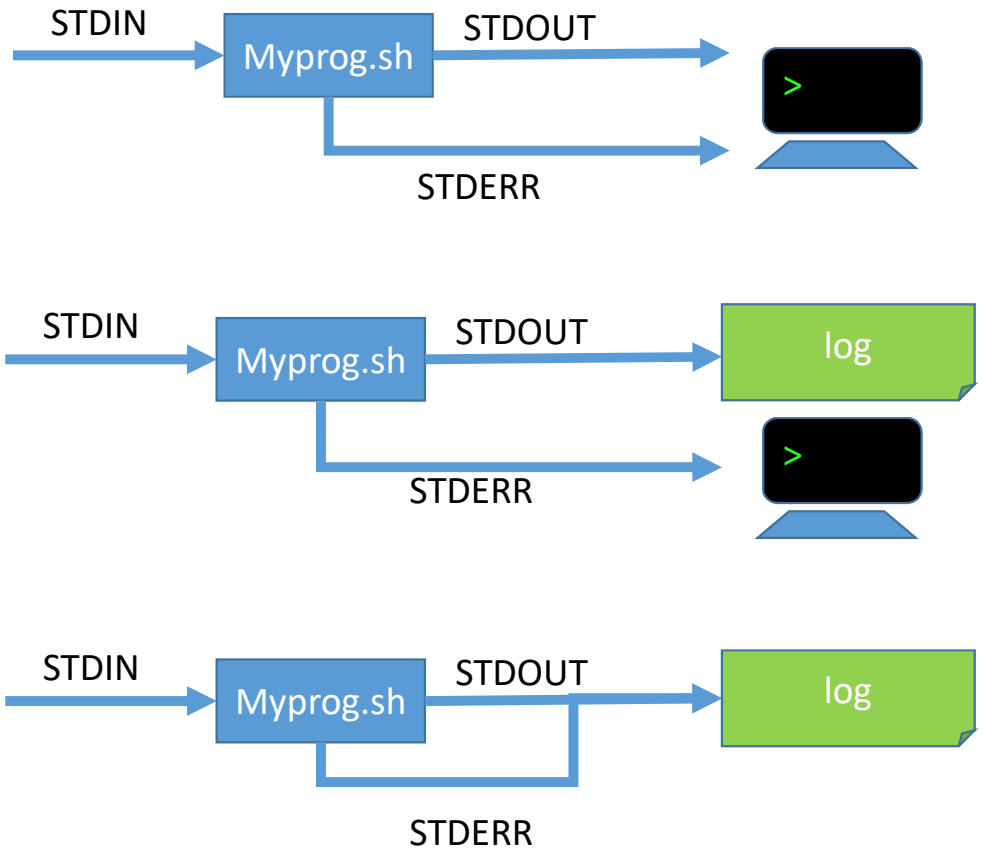





```
./myprog.sh 2>&1 > log
This is something for STDERR

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




← How do we get this instead?
```

# What if we 2>&1 but THEN change 1>…




```
./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?**

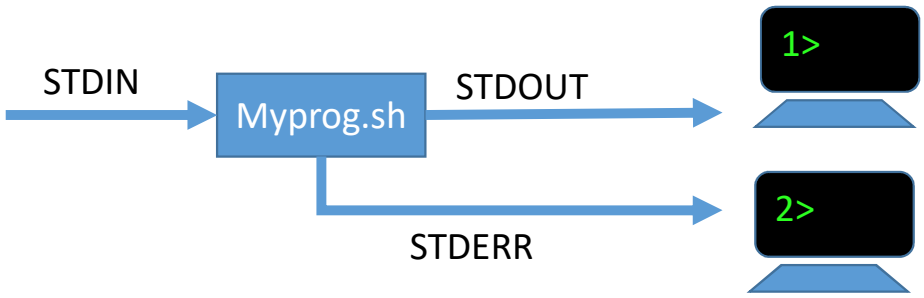
```
./myprog.sh > log 2>&1
```

Topic

# M3T1.4
# Bash Redirections – Swapping STDOUT & STDERR

https://youtu.be/0kY9nFO1078

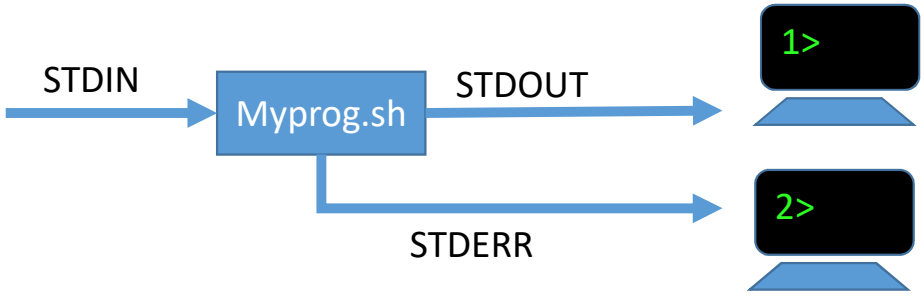
# Challenge: how to swap STDOUT / STDERR

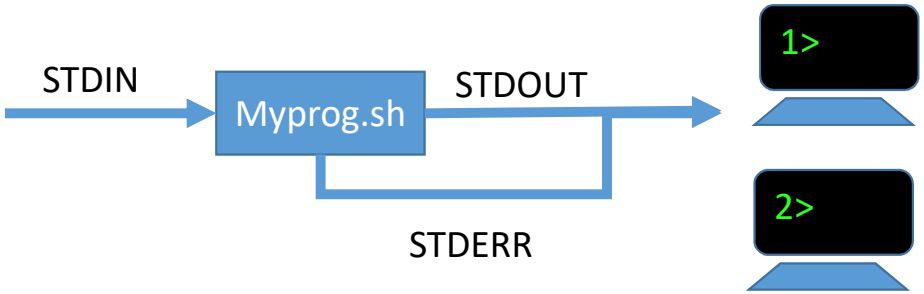



Let's start by only using what we learned so far

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

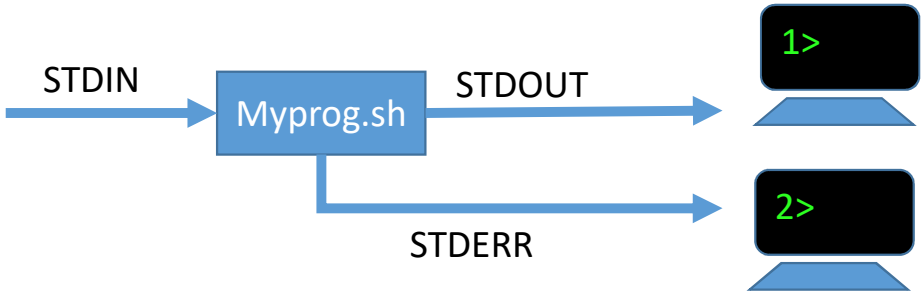
# Trying 2>&1

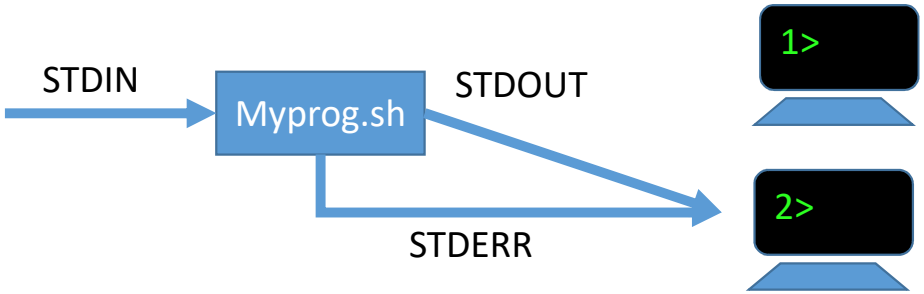



We lost STDERR target, whether it was the console or a file or a pipe…

# Trying 1>&2




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

# Problem is reminiscent of…

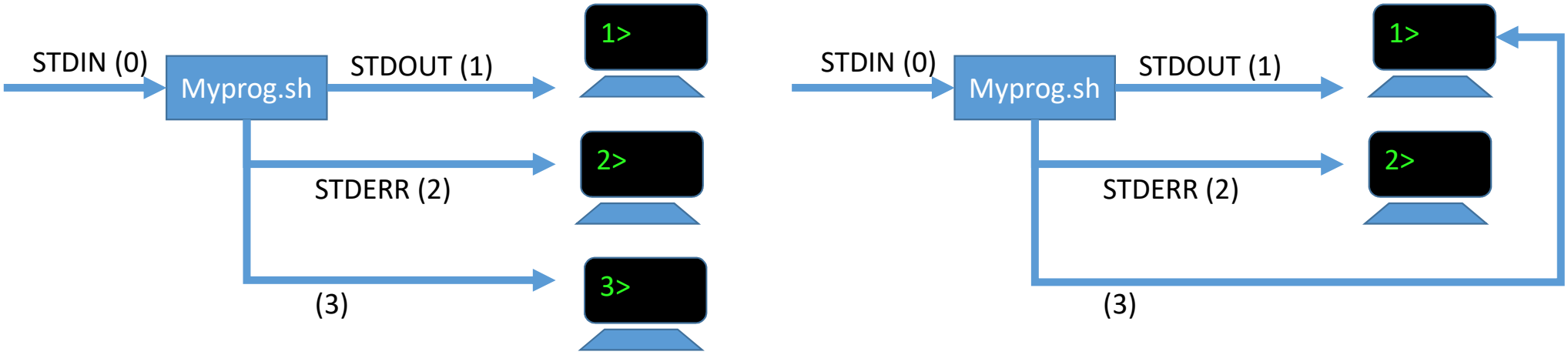
- Given two variables x and y, swap their contents.

- 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)

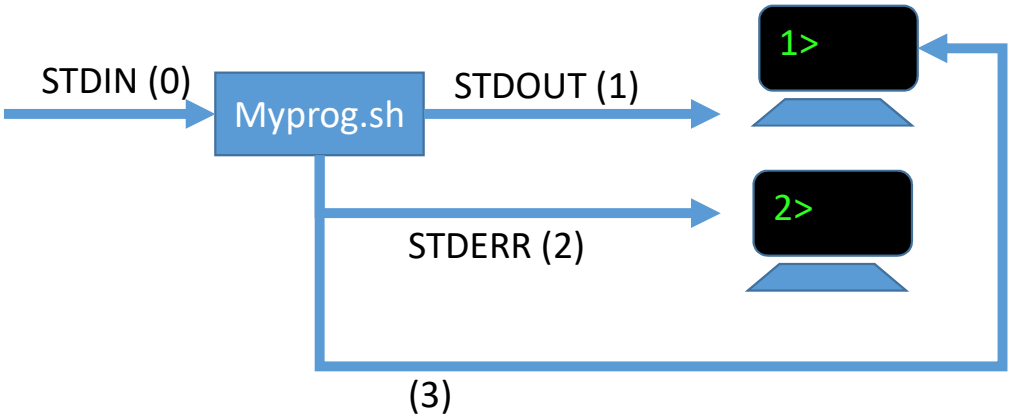
- 0, 1, and 2 are indexes in the file descriptors table
- There is a file descriptor at index 3 that we could use as TMP

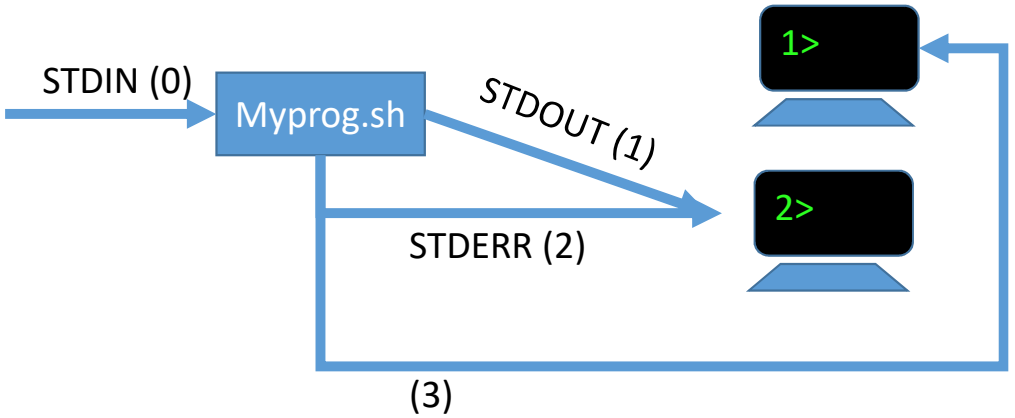



- 3>&1 → saves FD #1 in FD #3

# 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

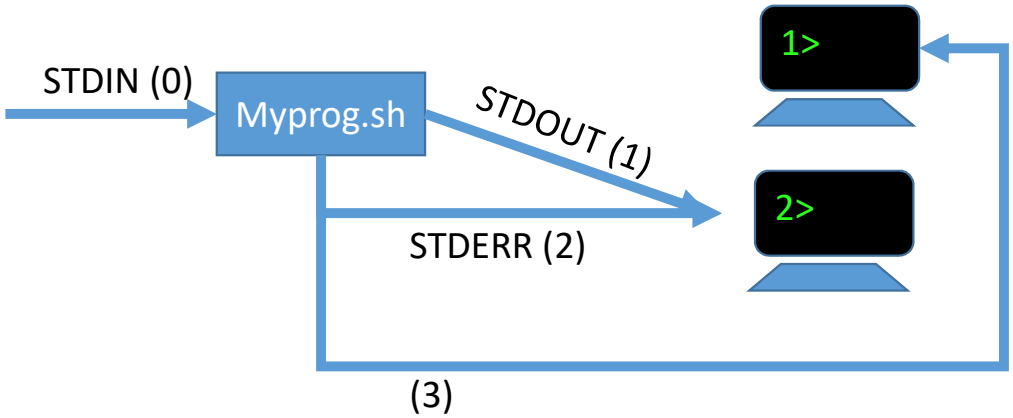


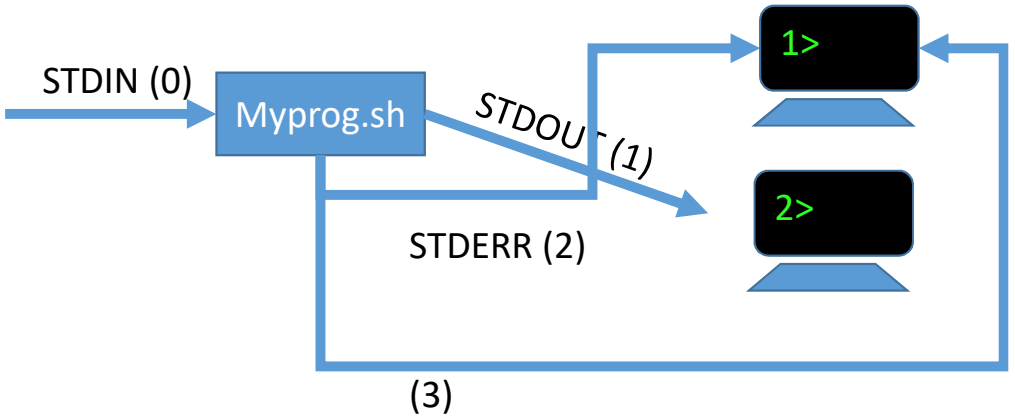




- 3>&1 1>&2

# 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…
```

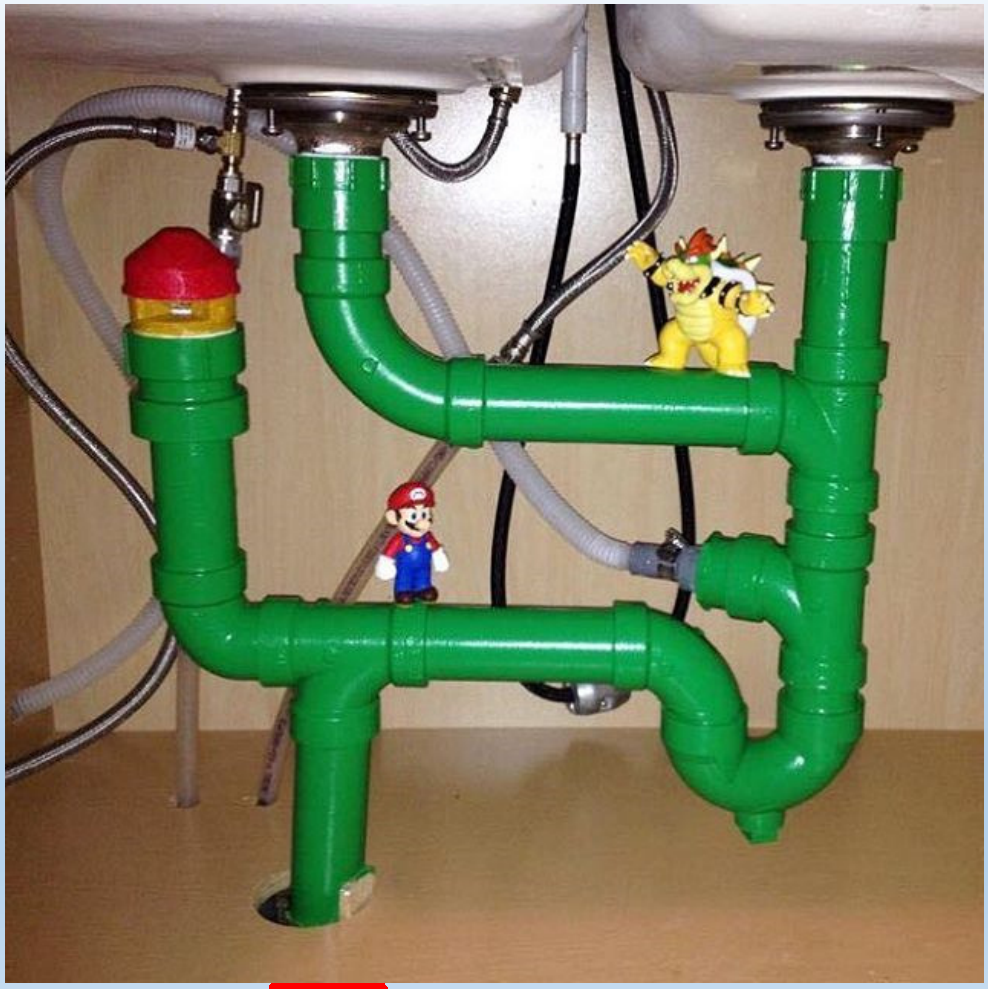
# Let's apply this to myprog.sh

- 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!

```
./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
```

Topic

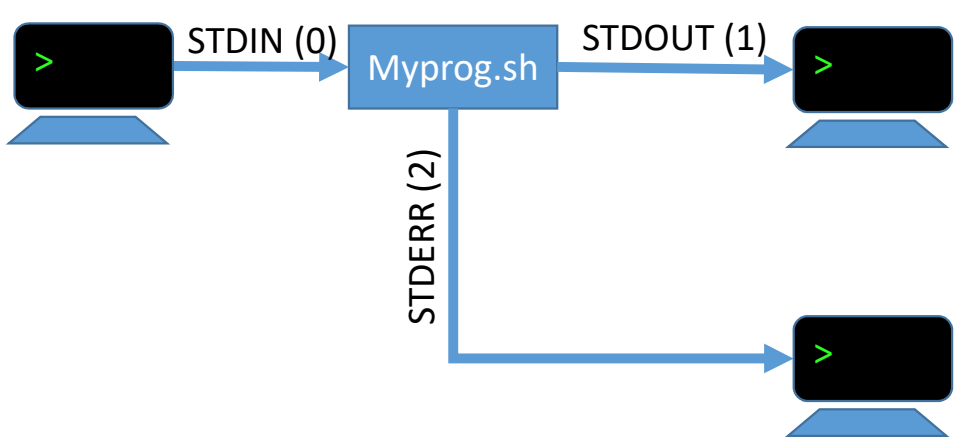
# M3T1.5 Bash Redirections – Piping

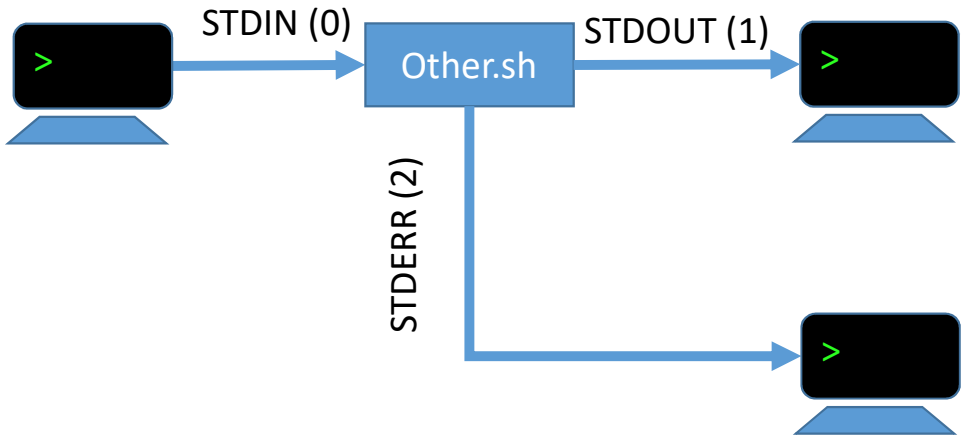


https://youtu.be/pPEX5q6odFI

# The story so far…

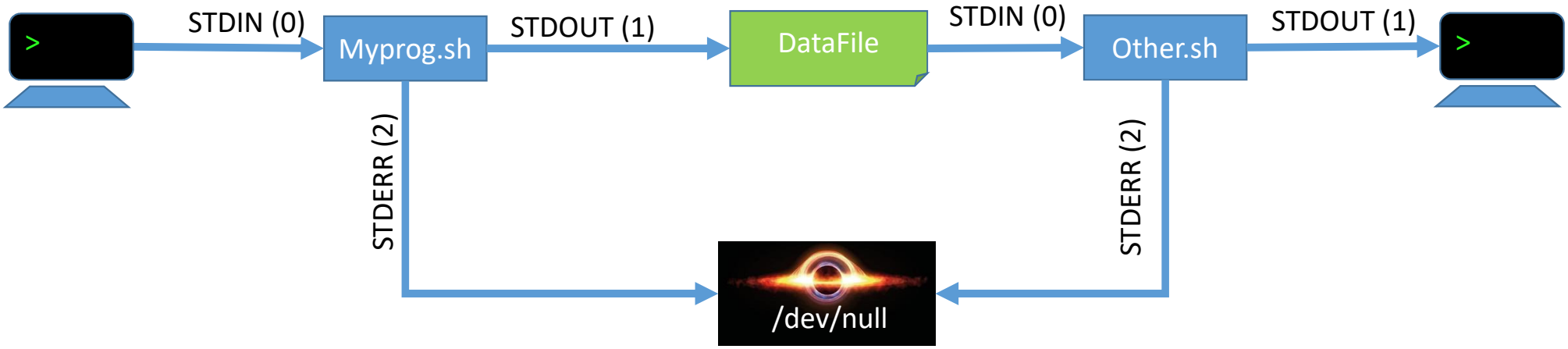
- Connecting FILES to STDOUT or STDERR
- What if we have two processes…



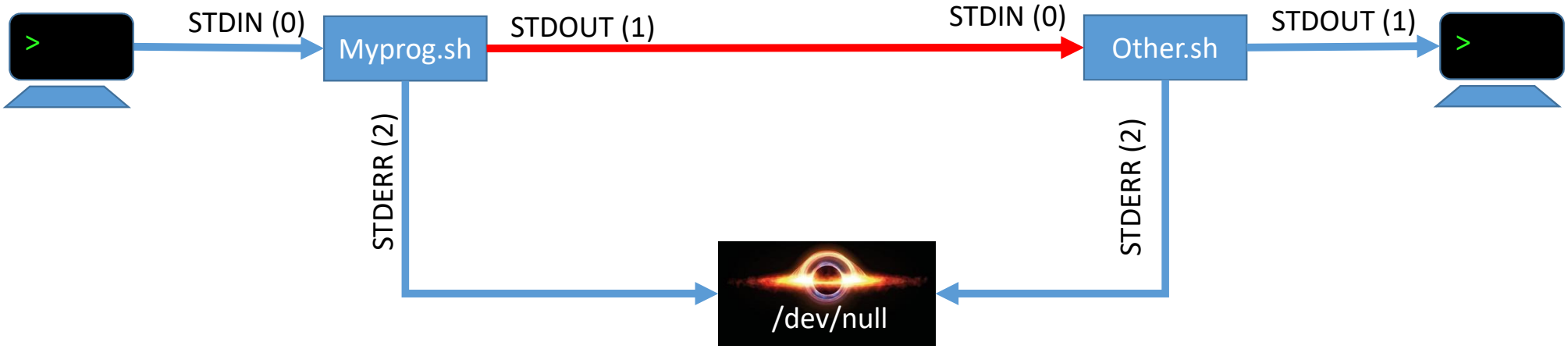
# The story so far…

- 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



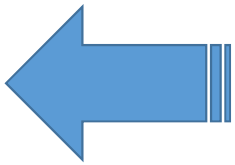
# What about we skip using DataFile?

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



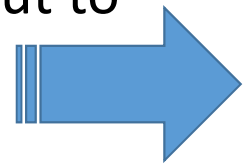
# Let's find some commands to illustrate this

```
cat
One
One
Two
Two
^D

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



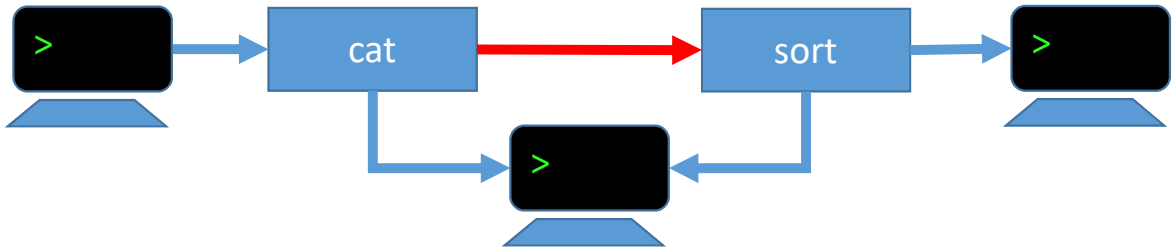
We need a command that reads from STDIN (kbd) and displays on STDOUT (screen)

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:









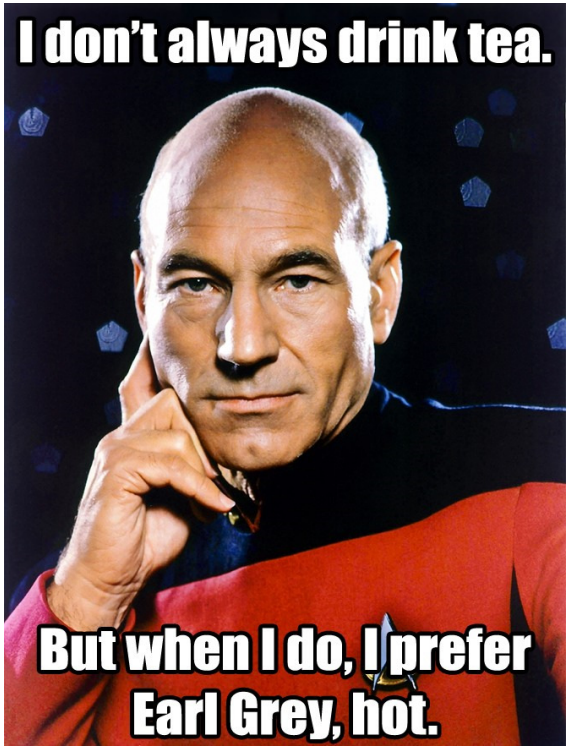


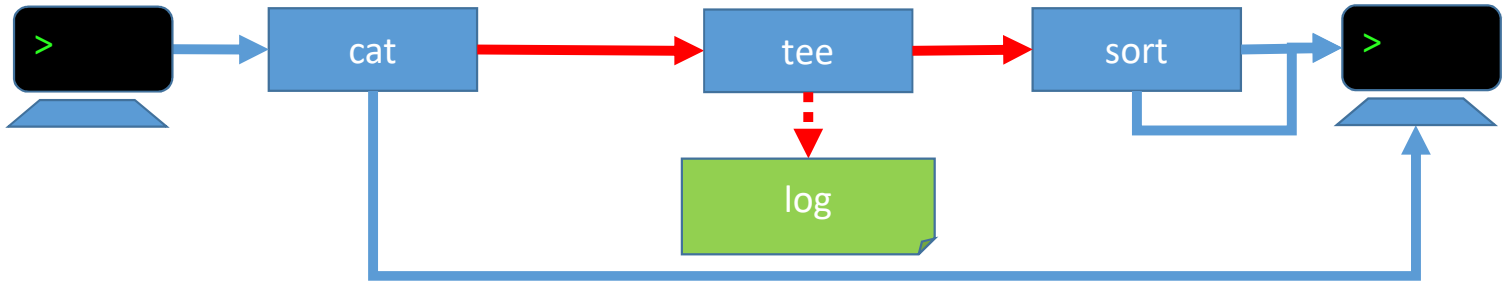

```
cat | sort
One
Two
Four
^D
Four
One
Two
```

# Introducing Tee. Earl Grey. Hot

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

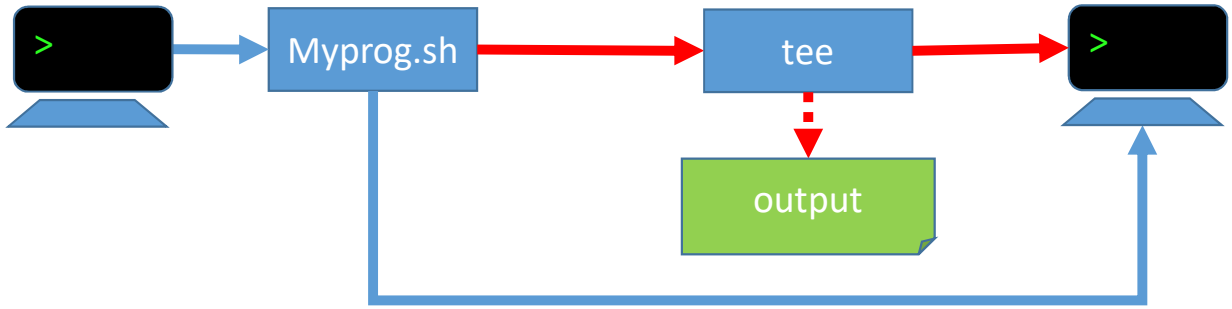
```
cat log
One
Two
Four
```



# 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
```











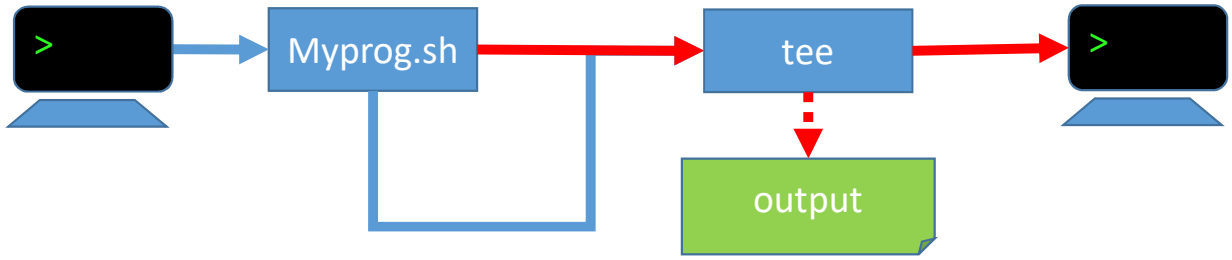

- This is what happens with just a regular |

# How to pipe both STDOUT and STDERR?

Not working on MacOS

```
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
```

- 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



# How do we make sure all ends 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