– General Linux 1 –
Use Streams, Pipes, and Redirects [3]

(Linux Professional Institute Certification)

[a]

```
   .~.        Prepared by Andrew Eager
   /V\
  // \\       geoffrey robertson
  @._.@       geoffrey@zip.com.au
```

$Id: gl1.103.4.slides.tex,v 1.2 2003/05/29 14:10:18 geoffr Exp $

(1.3) GNU and UNIX Commands [30]

1.103.1 Work on the command line [4]

1.103.2 Process text streams using filters [7]

1.103.3 Perform basic file management [2]

1.103.4 **Use streams, pipes, and redirects [3]**

1.103.5 Create, monitor, and kill processes [7]

1.103.6 Modify process execution priorities [2]

1.103.7 Search text files using regular expressions [3]

1.103.8 Perform basic file editing operations using vi [2]

Objective

Candidate should be able to redirect streams and connect them in order to efficiently process textual data. Tasks include redirecting standard input, standard output, and standard error, piping the output of one command to the input of another command, using the output of one command as arguments to another command and sending output to both stdout and a file.

Key files, terms, and utilities

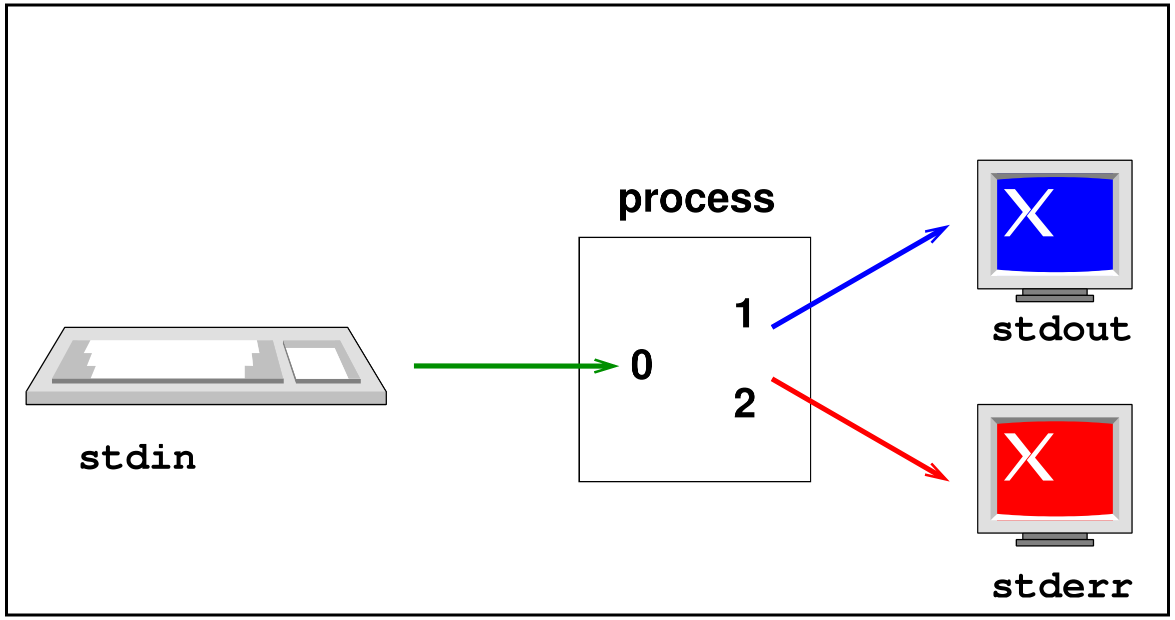
```
tee
xargs
<
<<
>
>>
|
``
```

Resources of interest

STDIN, STDOUT & STDERR

- When a process is run it needs 3 things:
  - An input device (ie a keyboard)
  - An output device (ie a screen)
  - An error device - somewhere to send critical errors (normally the screen)
- Every process has 3 *file descriptors*
  - fd 0 is for input
  - fd 1 is for normal output
  - fd 2 is for error/abnormal output
- By default these devices all default to your current tty

Default File Descriptor Assignments



- fd 0 == stdin (keyboard)
- fd 1 == stdout (screen)
- fd 2 == stderr (screen)

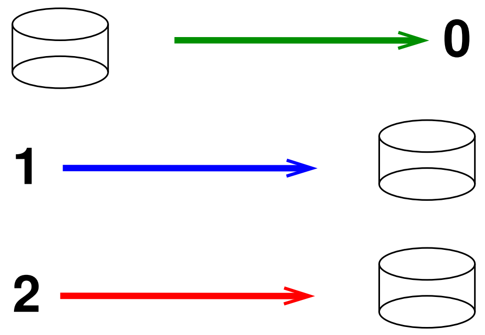
Redirection & Duplication Operators

- There are 3 operators used for redirection:
  - File redirects: ($<$, $>$and $>>$operators)
  - Pipelines ($|$operator)
  - File descriptor duplication ($>$& operator)

File Redirect Operators

Each of the 3 file descriptors can be redirected to/from files as follows:

Operator	Action
<	From file to fd 0
>	From fd 1 to file
2>	From fd 2 to file

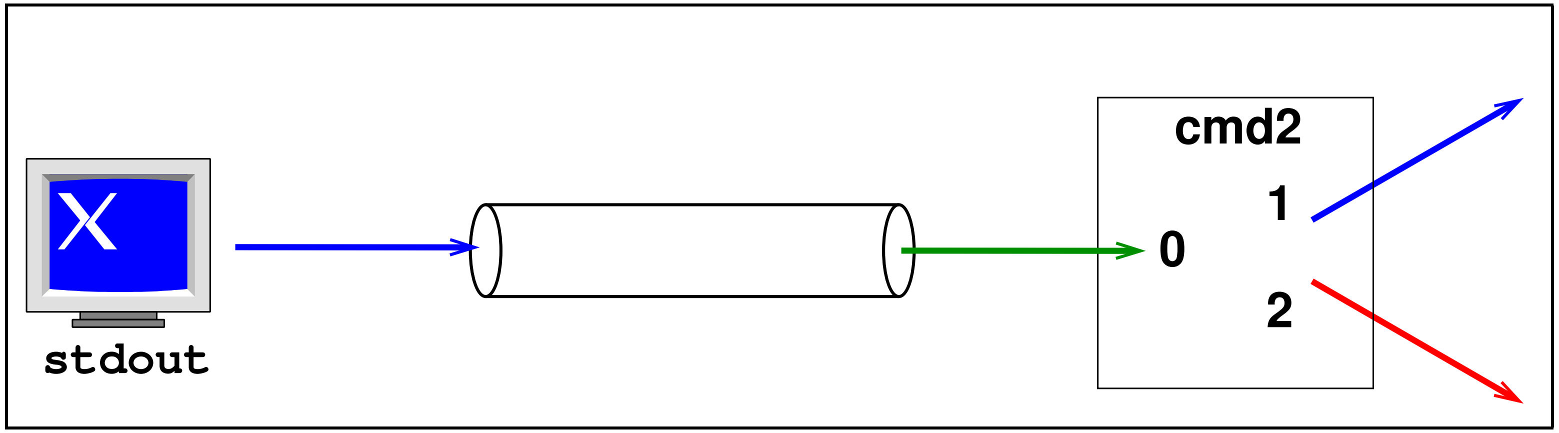


(Replace > with >> to append to a file)

Note that the redirect operators work with the file descriptors (0, 1 or 2) and *not* with the physical device itself (/dev/tty).

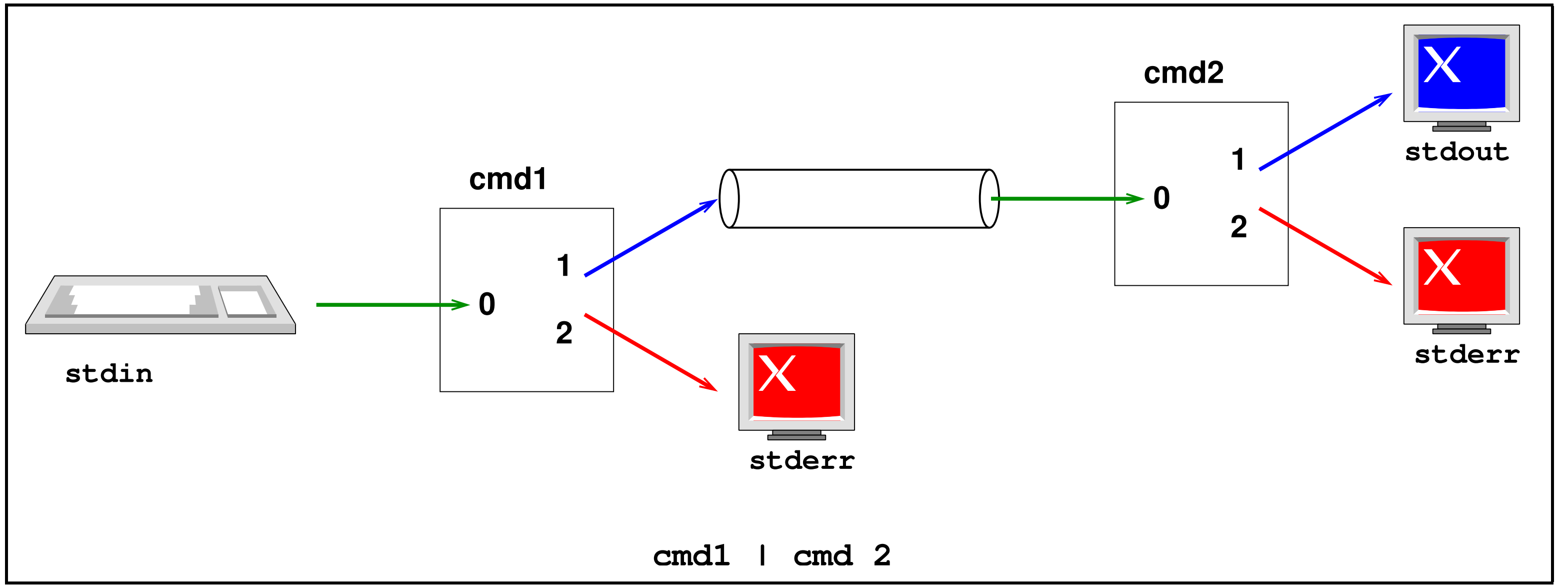
Pipeline Redirect Operator

Consider the command `cmd1 | cmd2`. The pipe operator takes data sent to stdout by cmd1 and sends it to the input file descriptor (fd 0) of cmd2:



Note that the pipe connects stdout (which may or may not be associated with fd 1) to fd 0 of the next command.

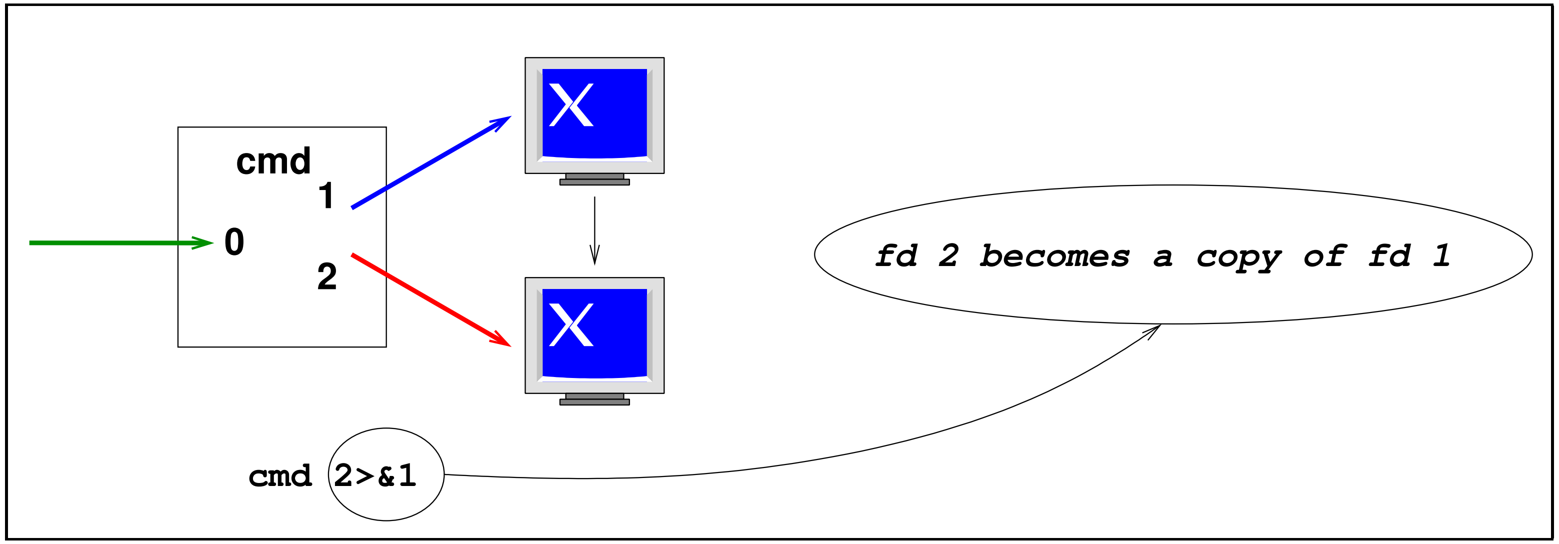
Example of the Pipeline Operator



Piping output of cmd1 into input of cmd2

File Descriptor Duplication Operator

A file descriptor can be made to be a copy of another descriptor. Consider the command `cmd 2>&1`. This will make fd 2 become a copy of fd 1.



Duplicating a file descriptor

Examples of File Output Redirection

For the following examples, we use the two example files `good` and `nofile` created thus:

```
$ echo "This is good" > good ↩
```

```
$ rm nofile ↩
```

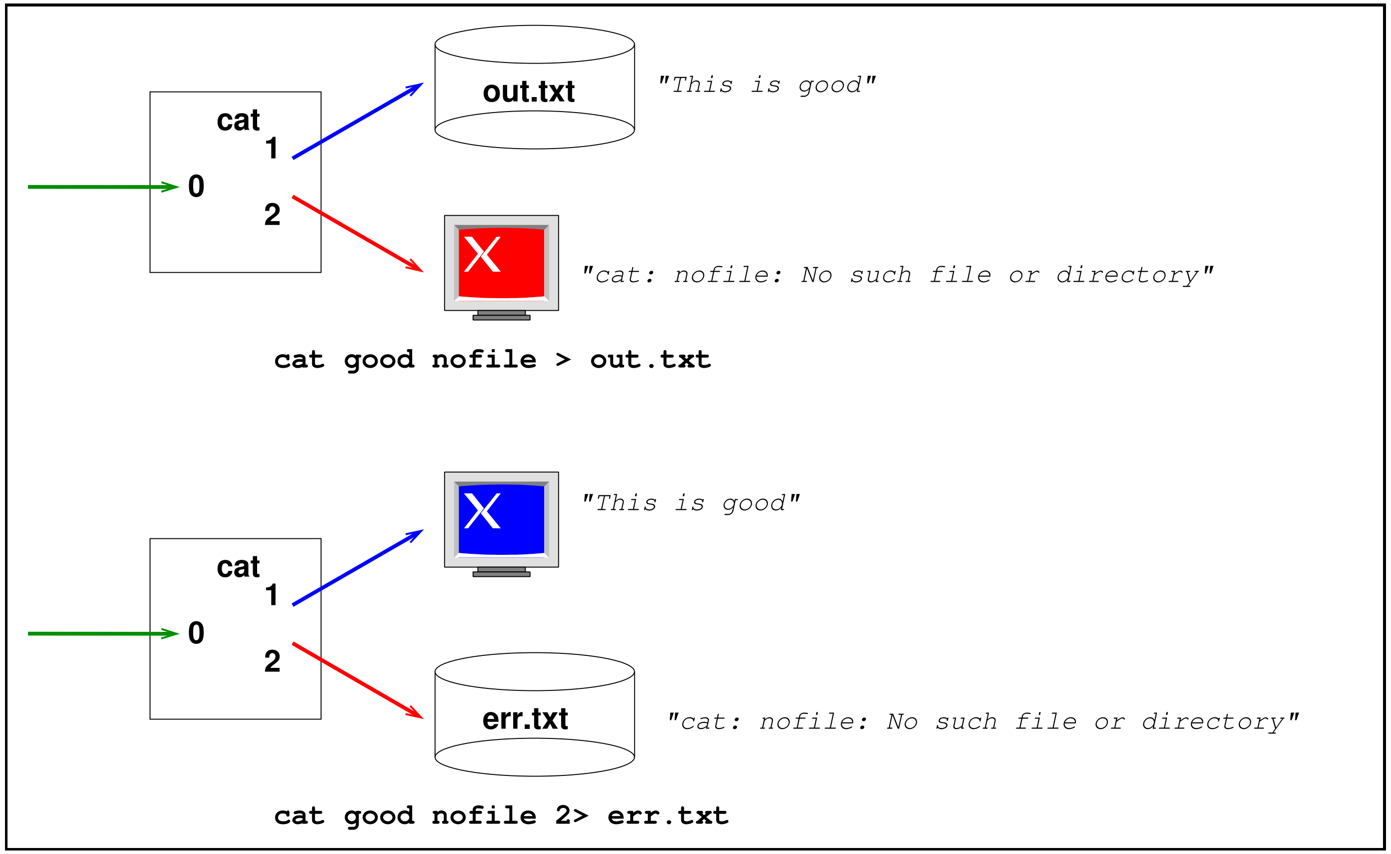
And to test what output is going where, we use the following command line:

```
$ cat good nofile ↩
```

which will produce:

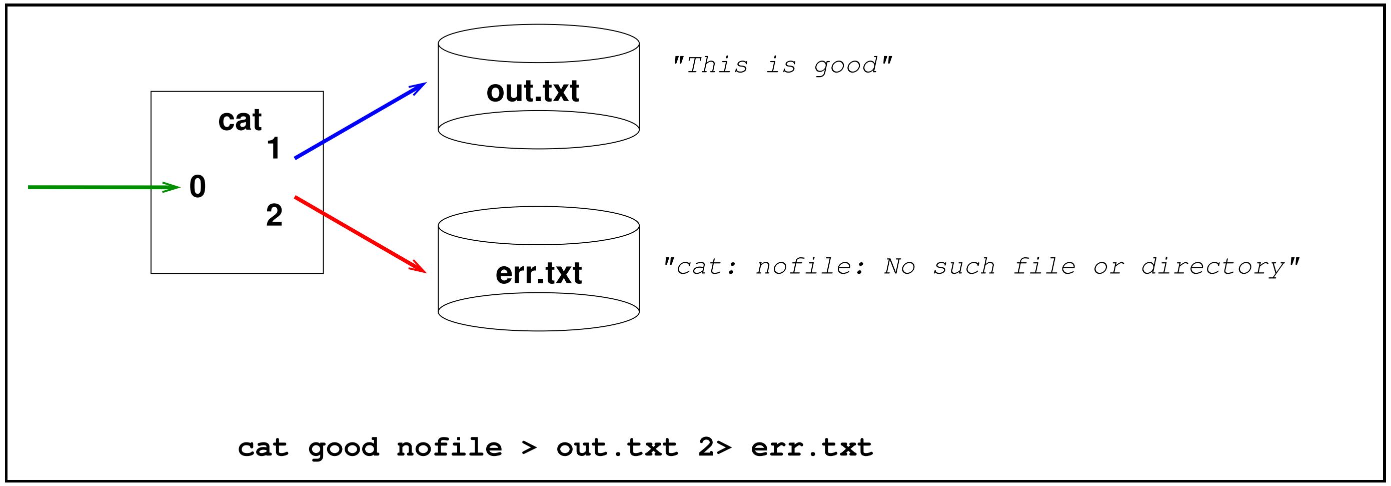
```
This is good                              (stdout)
cat: nofile: No such file or directory    (stderr)
```

Standard File Output Redirection


cat
0
1
2
out.txt
"This is good"
"cat: nofile: No such file or directory"
cat good nofile > out.txt
cat
0
1
2
"This is good"
err.txt
"cat: nofile: No such file or directory"
cat good nofile 2> err.txt

Redirecting stdout & stderr to different files

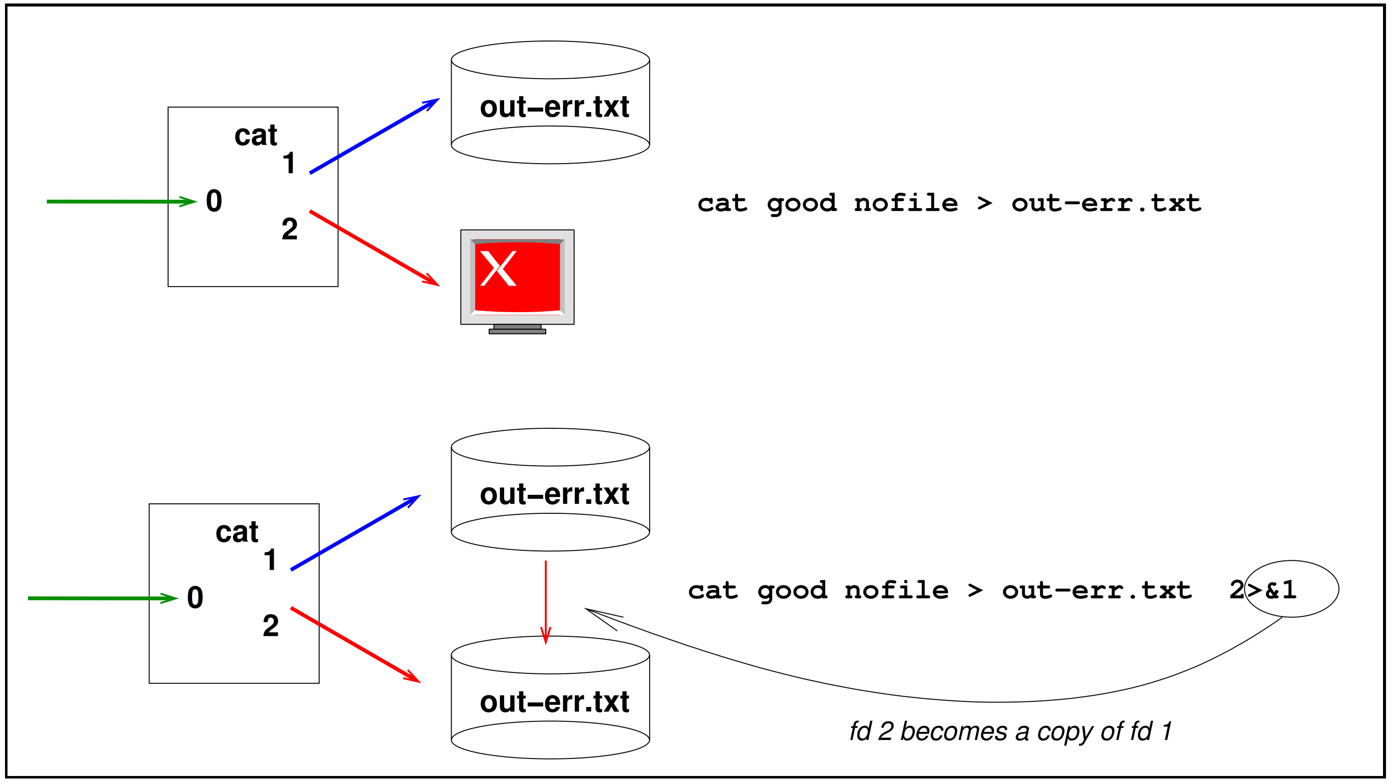
Output Redirection - Two at once



```
cat good nofile > out.txt 2> err.txt
```

Redirecting stdout & stderr at the same time

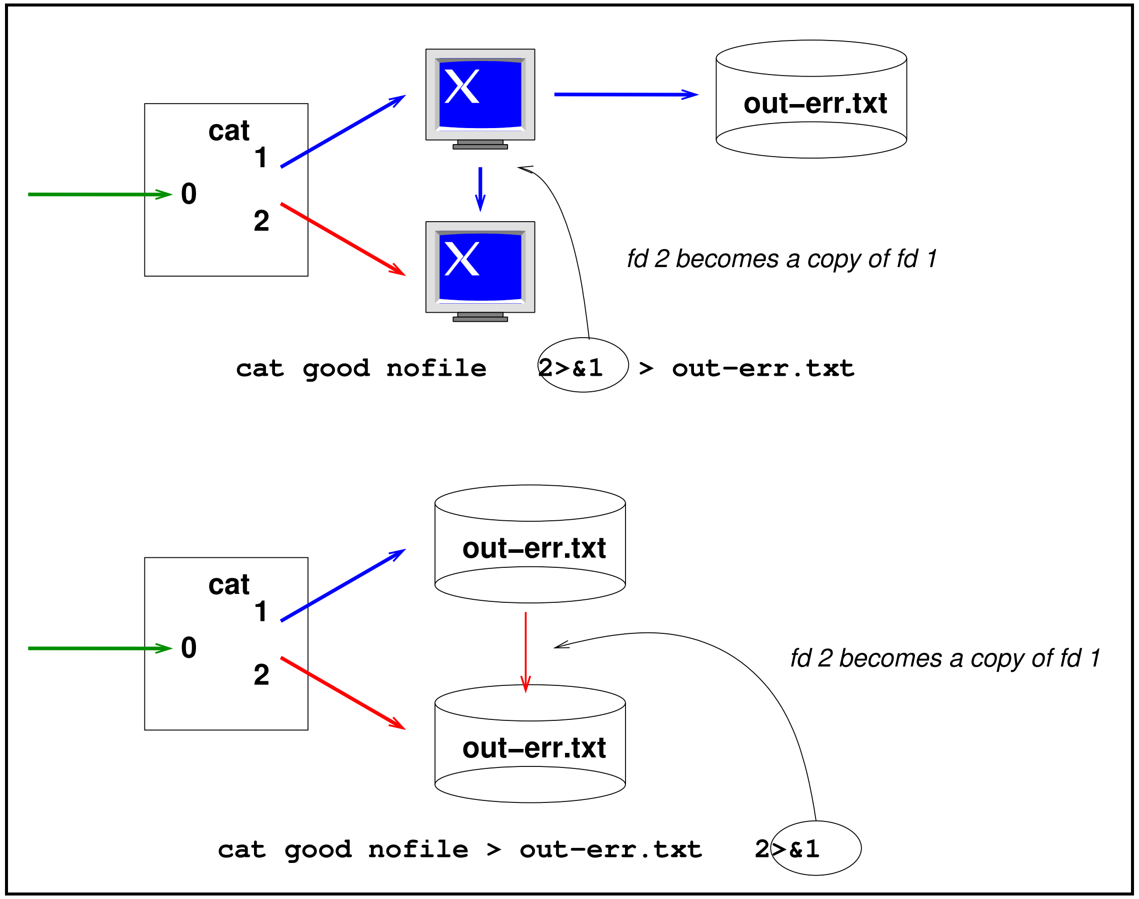
Redirecting stdout & stderr



cat
1
0
2
out–err.txt
X
cat good nofile > out–err.txt
cat
1
0
2
out–err.txt
out–err.txt
cat good nofile > out–err.txt 2>&1
fd 2 becomes a copy of fd 1

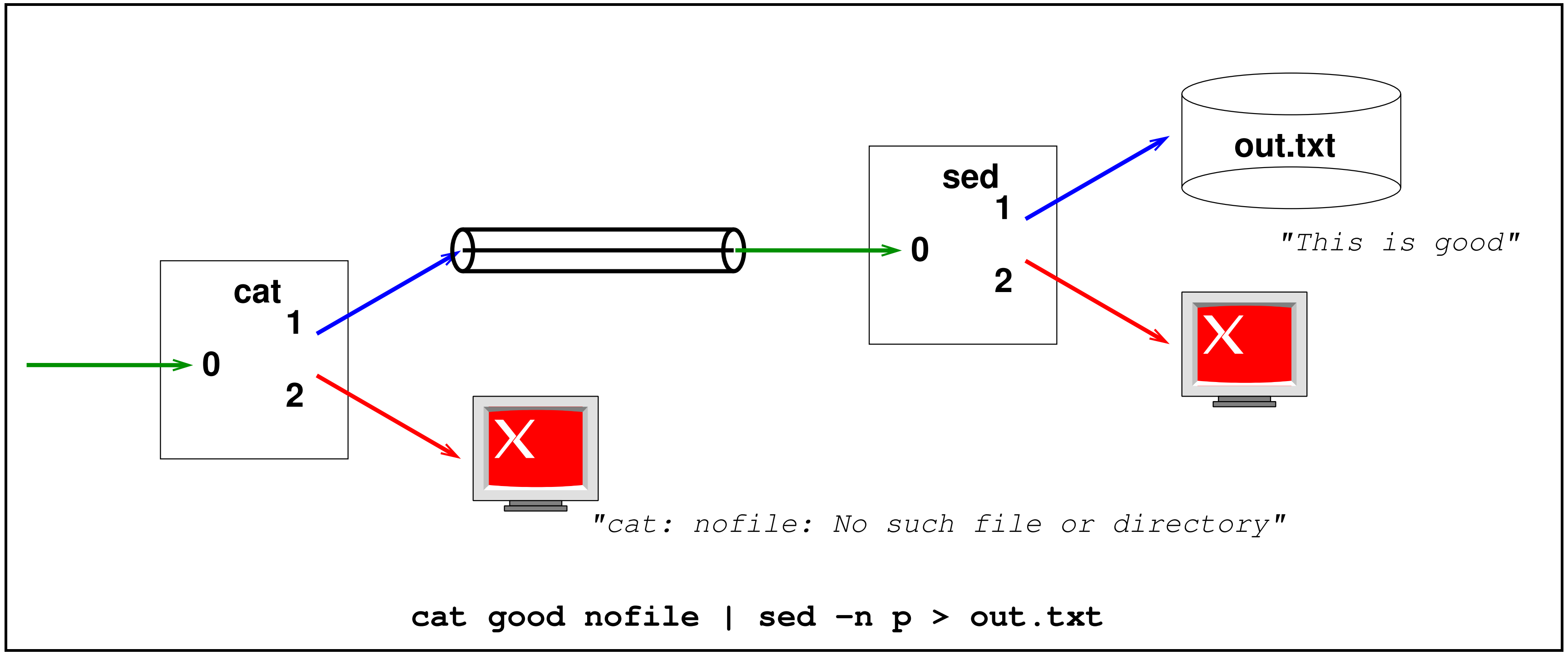
Redirecting stdout & stderr to the same file

Duplicating before & after redirection


cat
1
0
2
out–err.txt
fd 2 becomes a copy of fd 1
cat good nofile 2>&1 > out–err.txt
cat
1
0
2
out–err.txt
out–err.txt
fd 2 becomes a copy of fd 1
cat good nofile > out–err.txt 2>&1

Just when you duplicate the fd is significant

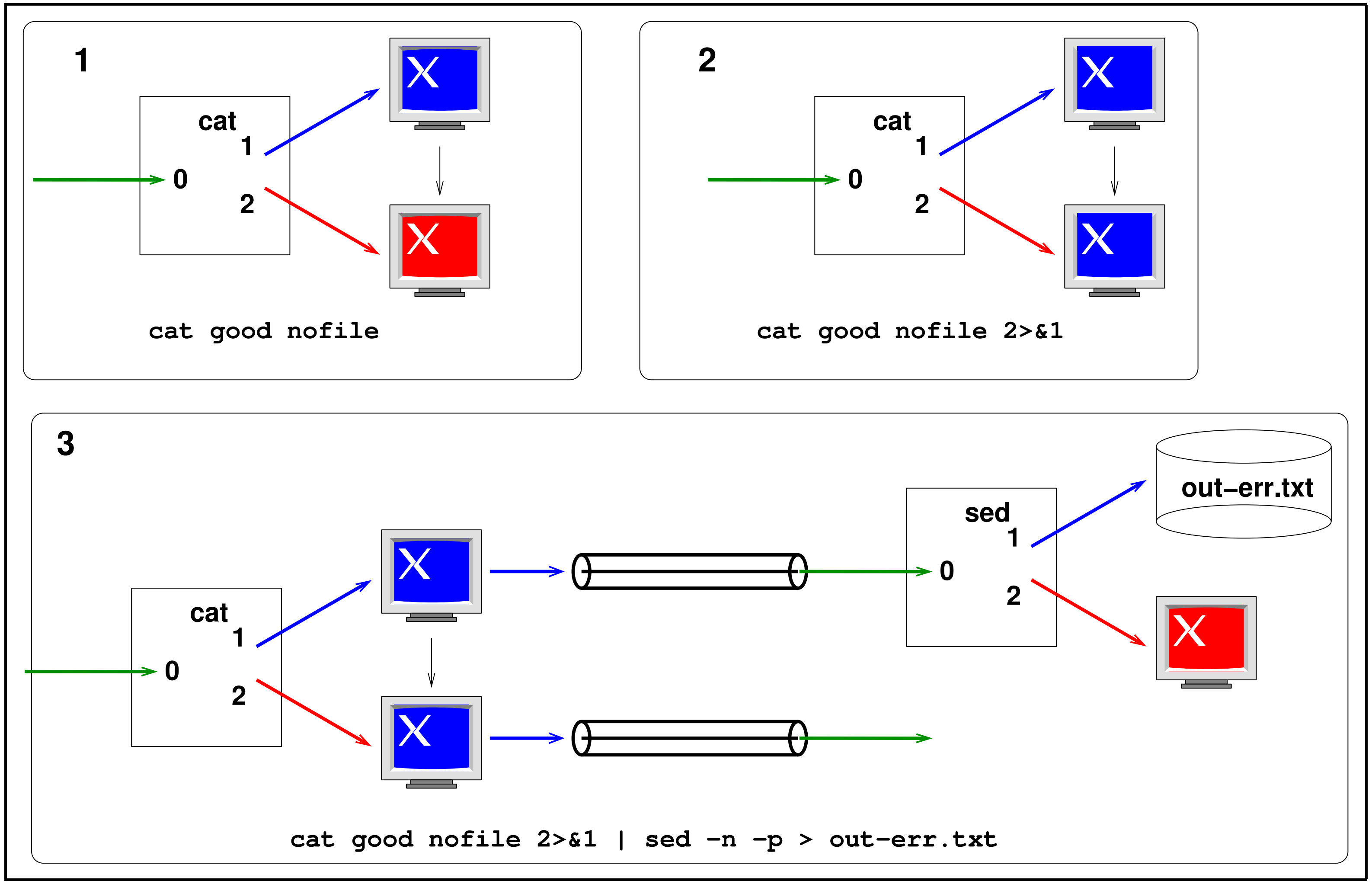
Piping stdout to stdin



```
cat good nofile | sed -n p > out.txt
```

Normal pipe from cat to sed

Piping stdout & stderr to stdin

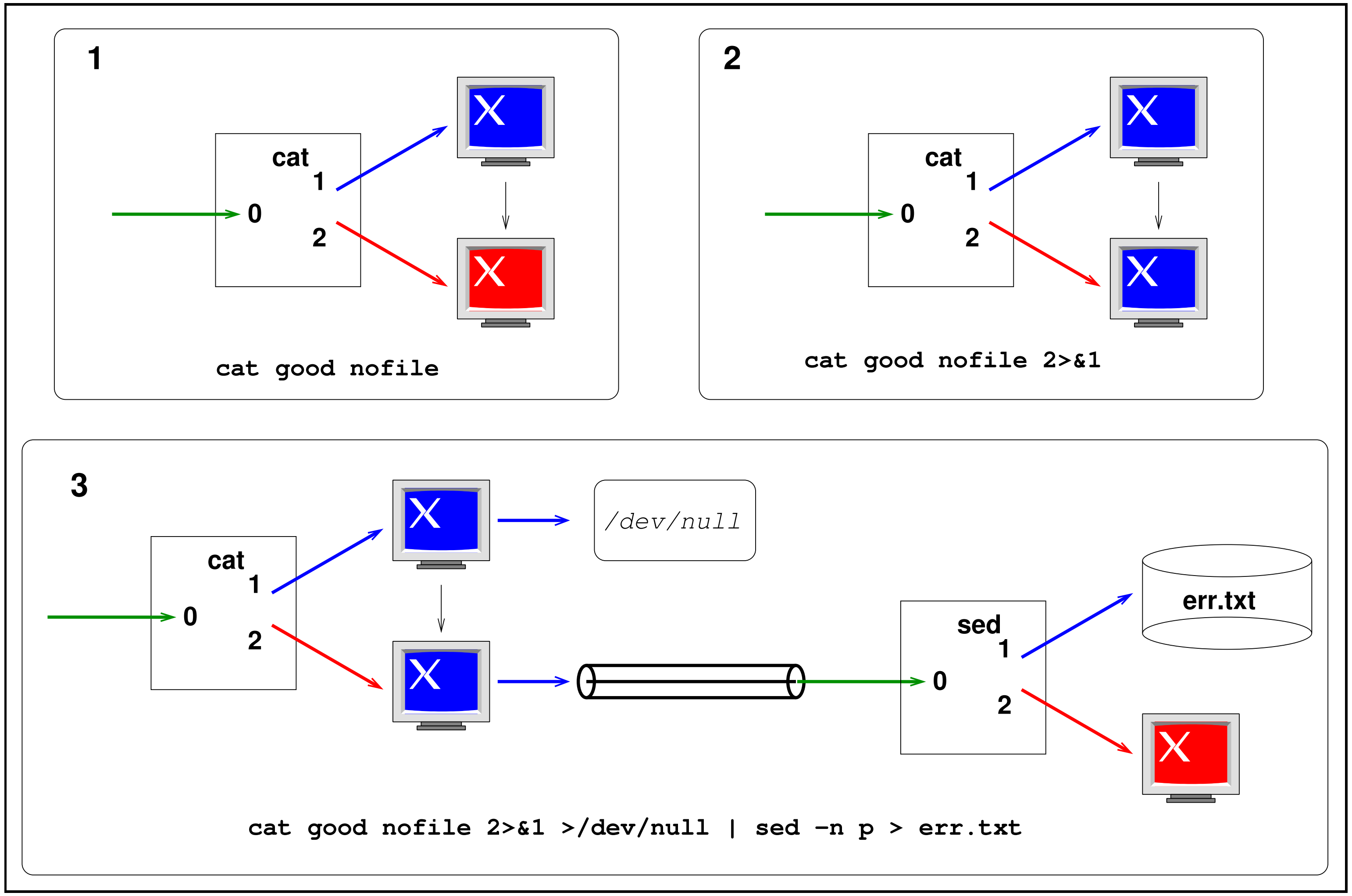


```
cat good nofile
```

```
cat good nofile 2>&1
```

```
cat good nofile 2>&1 | sed -n -p > out-err.txt
```

Piping stderr to stdin



```
cat good nofile
```

```
cat good nofile 2>&1
```

```
cat good nofile 2>&1 >/dev/null | sed -n p > err.txt
```

File Redirection Summary

- Redirect stdin from file:
  - $ **cat < input.txt** ↩
- Redirect stdout to file:
  - $ **cat good nofile > out.txt** ↩
- Redirect stderr to file:
  - $ **cat good nofile 2> err.txt** ↩
- Redirect stdout & stderr to file:
  - $ **cat good nofile > out-err.txt 2>&1** ↩

    OR
  - $ **cat good nofile 2> out-err.txt 1>&2** ↩

File Redirection Summary

To append to a file instead of overwriting, simply replace $>$with $>>$

- Redirect stdout to file (append):
  - $ **`cat good nofile >> out.txt`** $\hookleftarrow$
- Redirect stderr to file (append):
  - $ **`cat good nofile 2>> err.txt`** $\hookleftarrow$
- Redirect stdout & stderr to file (append):
  - $ **`cat good nofile >> out-err.txt 2>&1`** $\hookleftarrow$

    OR

  - $ **`cat good nofile 2>> out-err.txt 1>&2`** $\hookleftarrow$

Pipe Redirection Summary

- Pipe stdout to stdin:
  - $ **`cat good nofile | sed -n p`** ↩
- Pipe stdout & stderr to stdin
  - $ **`cat good nofile 2>&1 | sed -n p`** ↩
- Pipe stderr to stdin
  - $ **`cat good nofile 2>&1 >/dev/null | sed -n p`** ↩

A cool example - Swap stdout & stderr

In this example, we are going to swap stdout & stderr by using a temporary fd as a holding buffer:

If we execute the command using a normal pipe:

```
$ cat good nofile | sed -n -p > stdout.txt ↩
cat:  nofile:  No such file or directory
```

Now if we swap stdout & stderr:

```
$ cat good nofile 255>&1 1>&2 2>&255 | sed -n -p >
stdout.txt ↩
This is good
```

The End