Redirection

OS Practice CSC 420 Spring 2024

Marcus Birkenkrahe

March 12, 2024

README

- You can code along with the file tinyurl.com/4-redirection-org

  You can save it to an Org-mode file `4_redirection.org`
- The solutions and results herein were all obtained on a Lenovo laptop running Linux Mint 21.3 with 4 Intel Core i3-6006U CPUs.
- This lab is based on chapter 6 (pp. 49-59) of Shotts (2019).
- You can get the completed file from `tinyurl.com/5_redirection.org`

I/O redirection is the coolest command line feature
Important commands include:

- `cat` to concatenate files
- `sort` to sort lines of text
- `uniq` to report or omit repeated lines
- `grep` to print lines matching a pattern
- `wc` to print file newline, word, and byte counts
- `head` to output the first part of a file
- `tail` to output the last part of a file
- `tee` to read from `stdin` and write to `stdout` and files

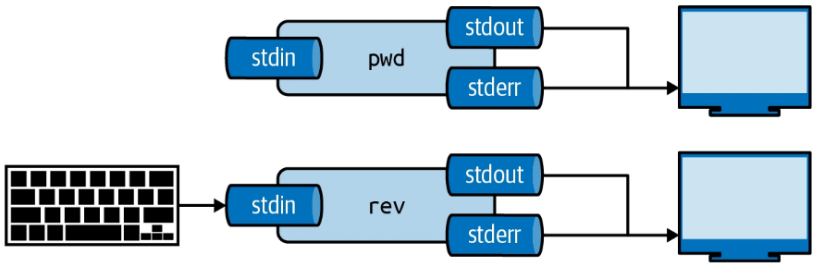

Figure 1: Source: cvskit - data science on the command line (2019)

Special device files

- `stdin`, `stdout` and `stderr` are special files.
- Do you remember where they are in the file tree?

```
ls -l /dev/* | grep std

lrwxrwxrwx  1 root root           15 Mar 10 16:35 /dev/stderr -> /proc/self/fd/2
lrwxrwxrwx  1 root root           15 Mar 10 16:35 /dev/stdin -> /proc/self/fd/0
lrwxrwxrwx  1 root root           15 Mar 10 16:35 /dev/stdout -> /proc/self/fd/1
```

- Execute the following commands in your REPL of choice. I'll do it in GNU Emacs Org Mode, of course.
- Do you remember what 'REPL' stands for?

  > Read-Eval-Print-Loop - an invention by Paul Graham for Lisp

- What does `rev` do?

```
whatis rev

rev (1)              - reverse lines characterwise

rev
all lines reversed
palindromes
Madam, I'm Adam
```

```
desrever senil lla
semordnilap
madA m'I ,madaM
```

Redirecting standard output

- It's often useful to store results in a file. What does the command in **??** do?

```
ls -l /usr/bin > ls-output.txt
```

- But what kind of file is created by default?

```
ls -l /usr/bin > kim
file kim

kim: ASCII text
```

- Let's look at the file from the outside first.

```
ls -lh ls-output.txt | head -n 10

-rw-rw-r-- 1 marcus marcus 182K Mar 12 12:06 ls-output.txt
```

- Let's look at the first and the last part of the file

```
head --lines=3 ls-output.txt  # or use -n 3
echo
tail --lines=2 ls-output.txt

total 389632
-rwxr-xr-x 1 root root       51648 Jan  8 08:56 [
-rwxr-xr-x 1 root root       18456 Feb  7  2021 411toppm

-rwxr-xr-x 1 root root          30 Mar 24  2022 zstdless
lrwxrwxrwx 1 root root           4 Jan 30 23:02 zstdmt -> zstd
```

- We redirect again, this time using a directory that does not exist. This results in an error - but why is it not sent to the file instead of `stdout`?

```
ls -l /bin/usr > ls-output.txt   # there is no /bin/usr, only /usr/bin
```

- What happened to the output file? It's empty!

```
ls -l ls-output.txt

-rw-rw-r-- 1 marcus marcus 0 Mar 12 12:06 ls-output.txt
```

- If you ever want to create a new empty file (instead of using the `touch` command), you can use `>` without origin (empty stdout).

```
rm ls-output.txt
> ls-output.txt
ls -l ls-output.txt
file ls-output.txt   # it's empty, it's not ASCII text

-rw-rw-r-- 1 marcus marcus 0 Mar 12 12:06 ls-output.txt
ls-output.txt: empty
```

- To append redirected output to a file instead of overwriting it from the beginning, use `>>`. Let's test this - compare with the initial size of the file (81K).

```
ls -l /usr/bin >> ls-output.txt
ls -l /usr/bin >> ls-output.txt
ls -lh ls-output.txt

-rw-rw-r-- 1 marcus marcus 364K Mar 12 12:06 ls-output.txt
```

Redirecting standard error

- To redirect standard error, use its **file descriptor.**
- File descriptors are internal stream references.
- We redirect stderr with `2>`. We use the command from before that tries to list a non-existing directory producing an error.

```
ls -l /bin/usr > ls-output.txt
ls -l /bin/usr 2> ls-error.txt
ls -lh ls-error.txt ls-output.txt
```

STREAM	FILE DESCRIPTOR
stdin	0
stdout	1
stderr	2

```
-rw-rw-r-- 1 marcus marcus 56 Mar 12 12:06 ls-error.txt
-rw-rw-r-- 1 marcus marcus  0 Mar 12 12:06 ls-output.txt
```

- What does it say?

```
cat ls-error.txt
```

```
ls: cannot access '/bin/usr': No such file or directory
```

- What if we want to redirect both `stdout` and `stderr` to the **same file** to capture all output in one place?
- Traditional method: use `2>&1` - "redirect `stderr` (2) AND `stdout` (1)": first we redirect `stdout` to a file, and then we redirect `stderr` (2) to `stdout` (1).

```
ls -l /bin/usr > ls-output.txt 2>&1  # double redirect
ls -lh ls-output.txt  # list stdout
cat ls-output.txt  # show output
```

```
-rw-rw-r-- 1 marcus marcus 56 Mar 12 12:06 ls-output.txt
ls: cannot access '/bin/usr': No such file or directory
```

  - The redirection of standard error must always occur **after** redirecting standard output. If the order is changed, `stderr` is directed to the screen instead. Try it yourself:
    1. create an empty file `output.txt` using >, redirect both `stdout` and `stderr` to it, then view and list the file:

```
> output.txt   # create empty file
ls -l /bin/usr > output.txt 2>&1  # stdout and stderr
cat output.txt  # view it
ls -l output.txt  # list it
```

```
ls: cannot access '/bin/usr': No such file or directory
-rw-rw-r-- 1 marcus marcus 56 Mar 12 12:06 output.txt
```

(a) Change the order of the redirection: first redirect stdout and `stderr`, then redirect to a file `output1.txt` with >.

```
ls -l /bin/usr 2>&1 > output1.txt
ls -lh output1.txt
```

```
ls: cannot access '/bin/usr': No such file or directory
-rw-rw-r-- 1 marcus marcus 0 Mar 12 12:06 output1.txt
```

* There is a more streamlined (but also more obscure) method for combined redirection with the single notation `&>`.

```
ls -l /bin/usr &> ls-output.txt
ls -l ls-output.txt
cat ls-output.txt
```

```
-rw-rw-r-- 1 marcus marcus 56 Mar 12 12:06 ls-output.txt
ls: cannot access '/bin/usr': No such file or directory
```

- Can you append stdout and stderr to a single file, too? Write and execute the command for appending with the single notation and the appending redirection operator!

```
ls -l /bin/usr &>> ls-output.txt
cat ls-output.txt
```

```
ls: cannot access '/bin/usr': No such file or directory
ls: cannot access '/bin/usr': No such file or directory
```

- *Silence is golden*: sometimes you just want to throw output away - like error or status messages. To do this, we redirect to a special file called `/dev/null`, also called the "bit bucket", or the "black hole".

  Write a command to redirect `stderr` from the error message to `/dev/null`, and then list the bit bucket file.

```
ls -l /bin/usr 2> /dev/null
ls -lh /dev/null
```

```
crw-rw-rw- 1 root root 1, 3 Mar 10 16:35 /dev/null
```

  `/dev/null` is a special character file (hence the letter `c` in the listing). The term is a Unix culture (see Wikipedia). More detail on the man page `null(7)` (`M-x man null`).

Redirecting standard input

- The `cat` command reads one or more files and copies them to standard output. To join more than one file, list the files to be joined after `cat`. If you don't specify a target, then the output will just be displayed as standard output.

  ```
  cat ls-output.txt ls-output.txt
  ```

  ```
  ls: cannot access '/bin/usr': No such file or directory
  ls: cannot access '/bin/usr': No such file or directory
  ls: cannot access '/bin/usr': No such file or directory
  ls: cannot access '/bin/usr': No such file or directory
  ```

- To have something to play with, let's split the `ls-output.txt` file. If your current file is empty or only contains one line, quickly fill it up by running the following code several times:

  ```
  ls -l /bin/usr &>> ls-output.txt
  ```

  that appends the error message to the same file.

- Find out how many lines your `ls-output.txt` file has!

  ```
  wc -l ls-output.txt
  ```

  ```
  2 ls-output.txt
  ```

- My file now has N lines. Use `split` to split it into N files of `1` line. Switch on `--verbose` to see what's happening. There should be as many files as you have lines in the file. Check out the man page for `split` to see how to use it.

  ```
  split ls-output.txt -l 1 --verbose
  wc -l x*
  ```

  ```
  creating file 'xaa'
  creating file 'xab'
    1 xaa
    1 xab
    1 xac
    1 xad
    4 total
  ```

- Now use `cat` to join the files back together and redirect the output into a file called `joined.txt`. Use a wildcard to identify the split files instead of writing their full names, and confirm the number of lines using `wc`.

```
cat x* > joined.txt
cat joined.txt
wc -l joined.txt
```

```
ls: cannot access '/bin/usr': No such file or directory
ls: cannot access '/bin/usr': No such file or directory
ls: cannot access '/bin/usr': No such file or directory
ls: cannot access '/bin/usr': No such file or directory
4 joined.txt
```

- What happens if you enter `cat` with no arguments? Try this on a system shell, in Emacs: `M-x shell`. You should find that `cat` just sits there waiting for input. When you enter anything, it's being mirrored back from `stdin` to `stdout` (your screen).

  In the terminal, enter `cat`, then enter the following text, then press `C-d`:

```
The quick brown fox jumped over the lazy dog.
```

- To create a file called `lazy-dog.txt`, enter:

```
cat > lazy-dog.txt
```

  Then enter the text followed by `C-d` (you have to press ENTER before):

```
The quick brown fox jumped over the lazy dog.
```

  You have just implemented the world's dumbest word processor! Check your results by viewing the file with `cat`.

```
cat lazy-dog.txt
```

```
The quick brown fox jumped
```

- You can also redirect standard input from the file `lazy-dog.txt` to `cat`. Do this in the following code block. If you get an error, think about what the shell sees.

```
cat < lazy-dog.txt
```

```
The quick brown fox jumped
```

> The command `lazy-dog.txt > cat` does not do the job: it tries to redirect a non-existing command into a file called `cat`. You have to redirect the `txt` file into `cat` from the right!

Pipelines

- Pipelines are used to perform complex operations on data. Remember this works because:

  1. every command is efficient at doing one specific job only
  2. commands can be put together with the pipe operator `|`

- Make a combined **list** of all the executable programs in `/bin` and `/usr/bin`, put them in **sorted** order, and **view** the resulting list. Remember that you can just fold the long output list by entering TAB on the `#+Results:` line.

```
ls /bin /usr/bin | sort | tail -n 10
```

```
zstd
zstd
zstdcat
zstdcat
zstdgrep
zstdgrep
zstdless
zstdless
zstdmt
zstdmt
```

The output of `ls` without the `sort` would have been two sorted lists, one for each directory. Check that by showing only the first 5 lines of the sorted, and of the unsorted pipeline. If you have difficulty keeping the output apart, you can put an `echo` in between the commands (generating an empty line).

```
ls /bin /usr/bin | sort | head -n 5
echo
ls /bin /usr/bin | head -n 5
```

```
[
[
411toppm
411toppm

/bin:
[
411toppm
7z
7za
```

- The redirection operator `>` is dangerous: it operates silently and will overwrite any system file if you use `sudo` privileges. This is a good way to destroy your OS. For example (don't try this!) - what would this command do?

```
cd /usr/bin
ls > less     # DONT DO THIS
```

- `uniq` is often used with `sort`. It accepts a sorted list of data from stdout or from a file and removes any duplicates.

  Add `uniq` after the `sort` to the pipe above. Replace the `less` command at the end by another command that allows you to compare the size of the files, but without using `ls`.

  Enter the pipeline above twice: once with and once without `unique`. Replace the `less` command at the end by a command that lets you compare the size of the output.

```
ls /bin /usr/bin | sort | wc -l
ls /bin /usr/bin | sort | uniq | wc -l

5631
2817
```

- In the next command, copy the code block **??**, and add the flag `-d` after `uniq` to only see the duplicates. Count the lines after each command with `wc`.

```
ls /bin /usr/bin | sort | wc -l
ls /bin /usr/bin | sort | uniq | wc -l
ls /bin /usr/bin | sort | uniq -d | wc -l

5631
2817
2814
```

- Another useful command is the pattern searching utility `grep`. It's most important flags are `-i` to make the search case insensitive, and `-v` to reverse the search and only print lines that do not conform to the pattern.

- Use `grep` to find all `zip` related commands in the output of our pipe from the block **??** (without the word count at the end). The beginning of the pipe is already in the block **??** below.

```
ls /bin /usr/bin | sort | uniq | grep zip

bunzip2
bzip2
bzip2recover
funzip
gpg-zip
gunzip
gzip
lzip
lzip-compressor
lzip-decompressor
```

```
lzip.lzip
mzip
p7zip
preunzip
prezip
prezip-bin
streamzip
unzip
unzipsfx
zip
zipcloak
zipdetails
zipgrep
zipinfo
zipnote
zipsplit
```

- How many programs in these directories are not zip-related?

  ```
  ls /bin /usr/bin | sort | uniq | grep -v zip | wc -l
  ```

  ```
  2791
  ```

- The utilities `head` and `tail` with the `-n N` option (`N` number of lines printed, also `--lines=N` as a long option) show beginning and end of files.

  `tail` has a real time option `-f` that allows you to monitor system logs. Run this command in the shell on `/var/log/syslog`.

  ```
  tail -f /var/log/syslog
  ```

  Using the `-f` option, `tail` continues to monitor the file, and when new lines appear, they appear on screen right away until you type CTRL-C. Check that by opening a new shell and typing something.

- Linux plumbing is rounded off by the command `tee` that creates a "tee" fitting on the pipe. It reads standard input and copies it to both standard output and to one or more files. In this way, the pipe can run on, and intermediate content can be captured, too.

  In the following command, we include `tee` in a pipe to capture the `ls` listing before filtering with `grep`.

```
ls /bin /usr/bin | tee ls.txt | grep zip | wc -l
wc -l ls.txt

52
5631 ls.txt
```

Linux is about Imagination

Windows is like a Game Boy. You go to the store and buy one all shiny new in the box. You take it home, turn it on, and play with it. Pretty graphics, cute sounds. After a while, though, you get tired of the game that came with it, so you go back to the store and buy another one. This cycle repeats over and over. Finally, you go back to the store and say to the person behind the counter: "I want a game that does this!" only to be told that no such game exists because there is no 'market demand' for it. Then you say, "but I only need to change this one thing!". The person behind the counter says you can't change it. The games are all sealed up in their cartridges. You discover that your toy is limited to the games that others have decided you need.

Linux, on the other hand, is like the world's largest Erector Set. You open it, and it's just a huge collection of parts. There's a lot of steel struts, screws, nuts, gears, pulleys, motors, and a few suggestions on what to build. So, you start to play with it. You build one of the suggestions and then another. After a while you discover that you have your own ideas of what to make. You don't ever have to go back to the store, as you already have everything you need. The Erector Set takes on the shape of your imagination. It does what you want.

Your choice of toys is, of course, a personal thing, so which toy would you find more satisfying? (William Shotts)