Redirection

Redirection

README

- 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
- `stdin`, `stdout` and `stderr` are special files
- `[ ]` Execute the following commands in your REPL of choice. I'll do it in GNU Emacs Org Mode, of course.

Redirecting standard output

- `[X]`

  It's often useful to store results in a file. What does the command in 1 do?

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

- `[X]`

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

  ```
  ls -lh ls-output.txt
  ```

- `[X]`

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

  ```
  head --lines=3 ls-output.txt
  tail --lines=2 ls-output.txt
  ```

- `[X]`

  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
  ```

- [X]

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

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

- [X]

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

  ```
  > ls-output.txt
  ls -l ls-output.txt
  ```

- [X]

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

Redirecting standard error

- [X] To redirect standard error, use its **file descriptor**
- [X]

  File descriptors are internal steam references.

STREAM	FILE DESCRIPTOR
stdin	0
stdout	1
stderr	2

- [X]

  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 2> ls-error.txt
ls -lh ls-error.txt
```

- [X] What if we want to redirect both stdout and stderr to the same file to capture all output in one place?
- [X]

Traditional method: use `2>&1` - "redirect stderr AND stdout": 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
ls -lh ls-output.txt
```

- `[ ]` 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 >, and redirect both stdout and stderr to it.

     ```
     ls -l /bin/usr >output.txt 2>&1
     cat output.txt
     ```

  2. 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
     ```

- `[ ]`

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

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

- `[ ]`

  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
  ```

- `[X]`

  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, then list the bit bucket file.

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

  `/dev/null` is a special character file (hence the letter `c` in the listing). The term is a Unix culture fix point (see Wikipedia).

Redirecting standard input

- `[X]`

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

- `[X]`

  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 several times:

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

  that appends the error message to the same file. My file now has three lines. Use `split` to split it into three 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.

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

- `[X]`

  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.

  ```
  cat x* > joined.txt
  cat 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
  ```

- `[X]`

  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 CTRL-D:

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

- `[X]`

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

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

  Then enter the text followed by CTRL-D:

  ```
  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 over the lazy dog.
  ```

- `[ ]`

  You can also redirect standard input from the keyboard to the file `lazy-dog.txt`. Do this now.

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

  If you get an error, think about what the shell sees. E.g. the command `lazy-dog.txt > cat` does not do the job: it tries to redirect a non-existing command into a file called `cat`.

Pipelines

- `[X]` Pipelines are used to perform complex operations on data. Remember this works because
  1. every command is efficient at doing one specific job
  2. commands can be put together with the `|` operator
- `[ ]`

  Make a combined list of all the executable programs in `/bin` and `/usr/bin`, put them in sorted order, 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 | less
  ```

  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 | tail -n 5
echo
ls /bin /usr/bin | tail -n 5
```

- `[ ]`

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

```
cd /usr/bin
ls > less
```

- `[ ]`

  `uniq` is often used with `sort`. It accepts a sorte 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
```

- `[X]`

  In the next command, copy 1, and add the flag `-d` to unique to only see the duplicates. Count the lines after each command.

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

- `[X]` 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.
- `[X]`

  Use `grep` to find all `zip` related commands in the output of our pipe from 1 (without the word count at the end). The beginning of the pipe is already in the block 1 below.

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

- `[X]`

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

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

- `[X]`

  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.

  ```
  tail -f /var/log/messages
  ```

  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.

- `[ ]`

  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
  ```

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)

Author: Redirection

Created: 2022-03-08 Tue 09:15

Validate