Bash (cygwin)

Redirecting the standard streams

Redirecting output

As we discussed in the previous lecture, the stream used for outputting something from a program went to a stream called standard out (stdout) and that was normally connected to the terminal the program was running.

But very often, we are not interested in watching all output from a program on the screen. Perhaps we want to save it in a file for using it later.

To do this, we tell the shell (bash) that we want to redirect this standard out stream to a file instead, using the > character:

```
$ pwd > the_path_to_here
```

Redirecting the output to a file

If we issue the command pwd, it prints the path to the current working directory on the screen (using stdout of course). But if we want it to write that information instead into a file (that will be created or overwritten) we did this:

```
$ pwd > the_path_to_here
```

If the file the_path_to_here didn't exist, bash created it for us, and then redirected the output from pwd as text into the file. If the file already existed, it was overwritten with the same text.

What can be redirected using ›

When you put a > after a command in bash, and then a file name, any output from the command that was written from the command will end up in the file with that name. But error messages are written to stderr, so they would still be written to the terminal window where you ran the command line.

The command ls writes the file it finds to standard out. But if we try to list the files in a directory that doesn't exist, it would complain and do so using the stderr stream, even if we use > to write any normal output to a file.

```
$ ls /this/is/incorrect > result_from_ls
ls: cannot access /this/is/incorrect: No such file or directory
```

What happened?

```
$ ls /this/is/incorrect > result_from_ls
ls: cannot access /this/is/incorrect: No such file or directory
```

The file result_from_ls was actually created, but is empty (0 bytes). But the error message from ls complaining that it couldn’t find the directory we tried to list, went to the terminal window, since we hadn’t redirected stderr (which ls used for its error message).

Appending to a file

If we don’t want to overwrite an existing file, we can use >> instead when redirecting. It then means, redirect but don’t overwrite - write at the end of the file instead.

```
$ echo "Line number 1" >> lines.txt
$ echo "Line number 2" >> lines.txt
$ cat lines.txt
Line number 1
Line number 2
```

Redirecting stderr

OK, but can we redirect also the stderr stream and create a log file with error messages? Yes we can:

```
$ ls /this/is/incorrect 2> errors_from_ls
```

Now, nothing was written to the terminal. So to redirect stderr we use 2> instead as the signal for “redirect stream number 2”. All streams have numbers, you see.

stdin: 0
stdout: 1
stderr: 2

Appending stderr

We can append stderr (rather than overwriting, write output to a new line at the end of an existing file):

```
$ rm errors_from_ls
$ ls /this/is/incorrect 2>> errors_from_ls
$ ls /this/is/incorrect 2>> errors_from_ls
$ cat errors_from_ls
ls: cannot access /this/is/incorrect: No such file or directory
ls: cannot access /this/is/incorrect: No such file or directory
```

The second time we ran the command, the error message ended up as a new line at the end of the file.

Redirecting both stdout and stderr

OK, but I want any normal result to go into one file, and all errors to another. No problem:

```
$ ls /this/is/incorrect > result_from_ls 2> errors_from_ls
```

Meaning: try to list the directory (or file) /this/is/incorrect. If you succeed, put the result in the file result_from_ls. If an error occurs, write it in the file errors_from_ls.

Let’s try it with both successes and failures

Let’s give ls two bad paths and two good ones, and write good results to a file and error messages to another file:

```
$ rm errors_from_ls result_from_ls
$ mkdir a b
$ echo "hello" > a/hello.txt
$ echo "goodbye" > b/goodbye.txt
$ ls a b c d 2> errors_from_ls > result_from_ls
$ cat errors_from_ls
ls: cannot access c: No such file or directory
ls: cannot access d: No such file or directory
$ cat result_from_ls
a:
hello.txt
b:
goodbye.txt
```

Redirecting stdin

We saw earlier that the calculator bc was capable of reading a mathematical expression from the user, and answer with the result.

If we don’t want to enter the expression manually, but rather that bc reads the expression from a file, we have to redirect stdin. We can tell bash that stdin no longer comes from the keyboard (the user typing it in), but stdin should be read from a file, using the < as signal for this type of redirection:

```
$ echo "9*9" > math1
$ bc < math1
81
```

Redirecting both stdin and stdout

Can we tell bc to read the math expression from a file, calculate it but write the result to a file? We'll we can tell bash that that's what we want. Remember, bc is not aware of such strange creatures as files. It only cares about the standard streams. But using redirection, we can achieve that like this:

```
$ bc < math1 > result1
$ cat result1
81
```