https://cs.senecac.on.ca/~david.ward/unx122/notes/Lab7\_Summary\_023.html

|  |
| --- |
| Seneca College of Applied Arts & Technology  UNX122 On-Line Lab #7  **1 Standard File Descriptors**  **2 Redirecting Standard Output  3 Redirecting Standard Input  4 Redirecting Standard Error  5 Connecting Commands with Pipes  6 Putting it All Together**   - Things to watch for  **7 Review Exercise**  **1. Standard File Descriptors**  When a Unix shell starts a command, it opens three connections for that  file. These three connections are called the 'standard file descriptors'. Many commands use these three connections for input and output.  These are the three standard file descriptors:       Number     Name              Short Name      **0          Standard Input    stdin      1          Standard Output   stdout      2          Standard Error    stderr**  The names, short names, and numbers of the standard file descriptors are important to know, so take a minute to memorize them now.  If you're using the shell interactively, then normally all three of the standard file descriptors are connected to your terminal (or telnet session). This means that when the program needs input, it gets it from the keyboard, and when it displays output or error messages, they appear on the screen.  Enter the 'date'command  **$  date Mon Mar 11 11:17:57 EST 2002**  Note that the date appeared on the screen.  This is because the standard output (stdout) of the date command was connected to the screen.  This connection between the stdout of the command and your screen is set up  automatically by the shell.  Now try using the '**cat**' command with no arguments.  IMPORTANT!: Once you enter '**cat**', you will be able to type text. Try entering several lines, pressing ENTER after each line. When you are done, press **Ctrl-D** on a line by itself to continue with the lab.  Notice that the 'cat' command just copied whatever you entered? That is because 'cat' with no arguments just takes whatever is entered on ***stdin*** (the keyboard) and copies it to ***stdout*** (the screen).  One of the features of Unix is that almost everything is treated as a file.  Enter the command 'who am i' to find out the file name for your terminal (look in the second column of output).  **$  who am i matrix!david.ward pts/0    Mar 11 11:09 (HSE-Toronto-ppp sympatico.ca**  '**who am i**' reported that your terminal device is pts/0 (second column of the output). That is a relative pathname based on the device directory /dev, so the full pathname for your terminal is **/dev/pts/0**  Check to make sure that file exists by getting a detailed listing of it.  **$  ls -l /dev/pts/0 crw--w----    1 david.wa tty      136,   0 Mar 11 12:15 /dev/pts/0**  Now use the 'file' command to see what kind of a file /dev/pts/0 really is.  **$  file /dev/pts/0 /dev/pts/0: character special (136/0)**  As you saw, /dev/pts/0 is a 'character special' file, which means that it is a device which accepts one character at a time and can send one character at a time.  It is possible to use the shell to connect the standard file descriptors to other files or devices, or even to other commands. The next sections of this lab discuss how this is done.  the file 'today' contained the date from the command 'date >today'  Next, try redirecting the output from 'cal' into a file named 'march'. |

|  |
| --- |
| **2 Redirecting Standard Output**  The standard output may be redirected with the symbol '>' followed by the name of the file which will contain the output.  You can think of the '>' character as being an arrow pointing from the command to the file, showing that the output from that command will go into the file.  Let's redirect the output of the 'date' command into the file 'today'.  **$  date >today**  Now display the contents of the 'today' file.  **$  cal > march      March 2002 Su Mo Tu We Th Fr Sa                 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31**  Let's redirect the output of the 'mesg' command into that same file ('march').  **mesg > march**  Now display the contents of the 'march' file.  **$  cat march is y**  Notice that the output from the 'mesg' command replaced the output from the 'cal' command!  This is called overwriting. The symbol '>>' will append (add) to a file instead of overwriting. You can think of the symbol as a double arrow from the command to the file, meaning 'add the output from the command to the END of the file'.  Try using '>>' to append the output from 'cal' to the file 'march' without overwriting what is already there.  **cal >> march**  Once again, display the contents of the 'march' file.  **$  cat march is y** **March 2002 Su Mo Tu We Th Fr Sa                 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31**  Good -- the output from 'cal' was appended to the information already in the 'march' file. |

|  |
| --- |
| **3 Redirecting Standard Input** You can use '<' to redirect standard input, just like you used '>' to redirect standard output.  You can also think of the '<' character as being an arrow pointing from the file to the command, meaning that the information will flow from the file into the input of the command.  First, let's create a little file that we can use for testing.  Let's put the current month's calendar into the file 'test'.  **$  cal > test**  The command 'write' takes its input from stdin, which is usually the terminal.  Use 'write' to send the contents of the file 'test' to yourself.  The command to write to yourself is 'write david.ward' -- add '<test' to make the command take its input from the file 'test'  **$  write david.ward < test  Message from david.ward@matrix on pts/1 at 14:30 ...      March 2002 Su Mo Tu We Th Fr Sa                 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 EOF**  You should have seen the message containing the calendar appear on your screen.  Another command that accepts input from stdin is 'mail'. You could use this feature to mail the 'test' file to yourself. Enter the command:  **$  mail david.ward@senecac.on.ca <test**  Some commands take their input from stdin if there is no file argument on the command-line.  For example, you can give a filename argument to the 'cat' command; but if you do not include a filename argument, then the input is taken from stdin (as you found out earlier).  This means that 'cat <test' and 'cat test' do the same thing.  Enter 'cat <test' now to prove this.  **$  cat <test** **March 2002** **Su Mo Tu We Th Fr Sa** **1  2** **3  4  5  6  7  8  9** **10 11 12 13 14 15 16** **17 18 19 20 21 22 23** **24 25 26 27 28 29 30** **31** |

|  |
| --- |
| **4. Redirecting Standard Error**   It is often useful to be able to separate error messages from a program's normal output.  For example, you may want to redirect the output from a command into a file, but still have any error messages go to the screen so that you can tell when something is wrong.  Let's intentionally create an error while redirecting the output.  **$  cal today >test cal: illegal year value: use 1-9999**  Notice that the error message appeared on the screen even though the output of the command was redirected. This is why standard error is separate from standard output.  To redirect standard error we use the same symbols as we use for redirecting standard output -- '>' and '>>' -- except that we add the file descriptor number for stderr (2) right in front of the symbol, like this:  **2>file     (overwrites) 2>>file    (appends)**  This time, let's intentionally create an error while redirecting stderr into a file.   Enter  cal today 2>errors  **$  cal today 2>errors $  cat errors cal: illegal year value: use 1-9999**  The 'errors' file contained the error message from the 'cal today' command.  For example, we can take the standard output of 'cal 2002' and connect it to the standard input of 'less' so that we can scroll the output on the screen.  Important: Remember that pressing q will exit from 'less'.  Try it out... |

|  |
| --- |
| **5. Connecting Commands with Pipes**                                                                                             In addition to connecting ***stdin***, ***stdout***, and ***stderr*** to files, it is possible to connect the ***stdout*** of one command to the ***stdin*** of another.  This is done using the pipe symbol: ' **|** '.  **cal 2002| less**  And the calendar is delevered one screen at a time (not shown here).  Notice that the commands are connected from left-to-right. The standard output of 'cal' was connected to the standard input of 'less'.    Let's try using 'head' to view the first 10 lines of the output from 'cal 2002'.  **$  cal 2002 | head                                2002         January               February                 March Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa        1  2  3  4  5                   1  2                   1  2  6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9 13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16 20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23 27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30                                               31**  Now let's add 'tail -5' to get just the last 5 lines of thost first 10 lines.  **$  cal 2002 | head | tail -5** **6  7  8  9 10 11 12    3  4  5  6  7  8  9    3  4  5  6  7  8  9** **13 14 15 16 17 18 19   10 11 12 13 14 15 16   10 11 12 13 14 15 16** **20 21 22 23 24 25 26   17 18 19 20 21 22 23   17 18 19 20 21 22 23** **27 28 29 30 31         24 25 26 27 28         24 25 26 27 28 29 30** **31**  Now we'll try some other examples...  Let's use 'ls -l' to get a detailed list of files in your home directory, and then use 'sort' to sort the list by file size. For the 'sort' command, we'll need the arguments '-k 5' to sort on the 5th column (file size) and '-n' to sort numerically instead of alphabetically.  **(large file)**  The files were sorted by size, with the largest at the bottom. But what if we only want to see the 2 largest files? Try the same command, but add 'tail -2' to see only the last two files listed.  $  ls -l|sort -n -k 5|tail -2 -rw-------    1 david.wa users       66048 Oct 19 16:47 UNX122\_w7\_p1.ppt -rw-------    1 david.wa users       88576 Oct 30 16:07 UNX122\_w8\_p1.ppt  We've been using pipes with text, but pipes can also be used with other types of data.  The School of Computer Studies logo is in the file '~unx122/cs.jpg'. It's compressed in jpeg format, which we can decode using 'djpeg'. We can then convert it from colour to grey with 'ppmtopgm', rotate it 90 degrees with 'pnmflip -r 90', and display it with 'display'.    Let's put these commands together:          **djpeg ~unx122/cs.jpg | ppmtopgm | pgmflip | display**  However, 'telnet' does not permit graphics to be displayed, so you will have to try this command in the lab sometime. |

|  |
| --- |
| **6 Putting it All Together**  Redirection and pipes are often used together, but it can be tricky getting the syntax right.  There are three important rules to remember:  (1) Redirection only works with files (>, >>, <).  (2) Pipes only work with commands (|).  (3) Data goes in the start of a pipe and flows out the end.  Redirection only works with files  One common mistake is to try to use the redirection symbols with commands. Remember that >, >>, < only work with files.  For example, this will work:          cal -y > today  These will not work:          cal -y > sort -n          sort -n < cal -y  Pipes only work with Commands  In the same way, the pipe symbol '|' can only be used to connect commands.  For example, this will work:          cal -y | sort -n  This will not work:          cal -y | /home/chris/outputfile  Data goes in the Start of a Pipe and Flows out the End  Pipes are written left-to-right. In this example:          a | b | c  The stdout of 'a' is connected to the stdin of 'b', and the stdout of 'b' is connected to the stdin of 'c'.  If you're going to combine file redirection with pipes, then any input redirection needs to be connected to 'a' and any output redirection needs to be connected to 'c'.  Data goes in the Start of a Pipe and Flows out the End  Therefore you could do something like this:          a <inputfile | b | c  But not:          a | b | c <inputfile  The problem with the second example is that the command shows the stdin of 'c' connected to both 'inputfile' and the stdout of 'b'.  Keep those three warnings in mind whenever you use redirection and pipes and you will avoid the most common mistakes!  (1) Redirection only works with files (>, >>, <).  (2) Pipes only work with commands (|).  (3) Data goes in the start of a pipe and flows out the end. |

|  |
| --- |
| **Question 1**: Redirect 'cal -y' into 'review1' **- Answer:** cal -y > review1   **Question 2:** Redirect the output of 'date' so that it is added to the end of the file 'review1' **- Answer:** date >> review1   **Question 3:** Using redirection, e-mail the contents of the file 'review1' to 'unx122@senecac.on.ca' **- Answer:** mail unx122@senecac.on.ca < review1   **Question 4:** Display a listing of the directory '/etc' one screen at a time. (Note: If you use 'less' or 'more', press 'q' when you want to quit). **- Answer:** ls -l /etc | less   **Question 5:** The file 'threelines' contains 3 lines of text. Display the middle line on the screen. You are not allowed to use the more command. **- Answer:** head -2 threelines | tail -1  **Question 6:** Display the date on the screen of the user 'kurt'. **- Answer:** date | write kurt  **Question 7:** Run the command 'mkdir old' and redirect any error messages into the file 'errors'. **- Answer:** mkdir old 2>errors  **Question 8:** Search the file 'data' for all of the lines that contain the pattern 'unx122' and put those lines in the file 'matches'. **- Answer:** grep unx122 data > matches  **Question 9:** Take the second line from the file 'data' and place it in the file 'second'. Place any error messages (from the last command in the pipeline) in the file 'errors'. **- Answer:** head -2 data | tail -1 > second 2>errors  **Question 10:** The file 'numbers' contains a list of numbers. Write a command to place the largest one of those numbers in the file 'largest' (there should be nothing else in that file). Do not use the 'head' command in your answer. **- Answer:** sort -n numbers | tail -1 > largest |