KIT104 ICT Architecture and Operating Systems
Tutorial Week 7

1. UNIX Shell (continued from week 6)

1.1 Redirection on UNIX Shell

When a command is executed on a UNIX shell, the default output destination is the terminal (computer screen). The default source for getting input is the keyboard. However, inputs and outputs can be redirected to come from or go to any disk files or some other device.

There are three destinations for standard output: the terminal (default), a file, and a pipe (introduced later).

`$ man who` - This displays the man pages of `who` on the screen
`$ man who > newfile` - This saves the man pages of `who` into a file named `newfile`.
`$ man ls >> newfile` - This appends the man pages of `ls` to the end of any existing content of `newfile`.

There are three sources for standard input: the keyboard (default), a file, and a pipe (introduced later).

`$ wc newfile` - the command `wc` opens the file `newfile`
`$ wc < newfile` - the shell opens the file `newfile`

The default destination of standard error is also the terminal:

```
$ cat  bar
```

In the above command line, if the file `bar` does not exist you would see the following error message:

```
cat: bar: No such file or directory
```

If you want to redirect this error message into a file you could try:

```
$ cat  bar  >  errorfile
```

But the error message still shows up on the terminal!

In UNIX, The inputs, outputs, and errors are actually streams of bytes treated as files, with each of them assigned a number called a file descriptor:
0 – standard input
1 – standard output
2 – standard error

`man ls > newfile` is the same as `man ls 1> newfile`
`wc < newfile` is the same as `wc 0< newfile`

To redirect error messages:

```
$ cat bar 2>  errorfile
```

Sometimes we need to manipulate a group of commands and then redirect them. For example:

```
$ (ls -x *.c; echo; cat *.c) > c_files_all.txt
```

This saves all C program files in a file, preceded by a multicolumn list of files acting as a table of contents. The `echo` command serves to insert a blank line between them.

A	Login to your alacritas account. Change into `kit104` directory. Remove any existing files or subdirectories under `kit104` (unless you really want to keep them).
B	Use `joe` or `nano` to create three text files which must be named as `file1`, `file2`, and `file3`, respectively. The content of `file1` must be: `Content of file1` (Note: press the `Enter` key at the end of the line before saving the content) The content of `file2` must be: `Content of file2` (Note: press the `Enter` key at the end of the line before saving the content) The content of `file3` must be: `Content of file3` (Note: press the `Enter` key at the end of the line before saving the content) Use `cat` to verify the file contents after they have been created.
C	Run the following command lines and think about the outputs: `$ ls` `$ cat file1 > files` `$ ls` `$ cat files` `$ cat file2 > files` `$ ls` `$ cat files` `$ cat file3 > files` `$ ls` `$ cat files` Have you seen that redirecting output using > erases any pre-existing content? erases pre-existing content Now let's try it again using a different approach to see how you can keep any pre-existing content of a file:

	`` $ ls $ cat file1 > files $ ls $ cat files $ cat file2 >> files $ ls $ cat files $ cat file3 >> files 追加 $ ls $ cat files ```
D	It is possible to redirect a group of commands using a single command line: ``` $ rm files $ ls $ (cat file1; cat file2; cat file3) > files 全部一起汇总 $ ls $ cat files ```
E	Next, let's try redirecting both input and output. First of all, let's slightly change the content of `files`: ``` $ rm files $ ls $ (cat file3; cat file2; cat file1) > files $ ls $ cat files $ sort < files 输入排序 $ cat files $ sort < files > files_sorted $ ls $ cat files_sorted ``` 输入排序再输出到files_sorted Can you see what has happened here?
F	Redirecting error messages is different. Ensure that you do not have a file named `foo` under the current directory. If you do, remove it first. ``` $ cat foo $ cat foo 2> errors $ ls $ cat errors ``` 输出错误信息

G	You have previously used the `find` utility to locate files on the system by specifying a set of criteria. You were told to get rid of the many "Permissions denied" error messages by redirecting them to the null device (`/dev/null`). What if you want to have a record of all the error messages generated from a `find` session? `$ find / -name "index.htm" 2> errors` (Note: this command line may take some time to run) `$ cat errors` 错误信息输出到文本
H	Remove the `errors` file.
I	What is the difference between the following command lines: `$ cat file1` `$ cat < file1` You see the same output from both command lines. The difference is this: In the first command line, the command `cat` opens the content of `file1`, but in the second command line, the shell opens the content of `file1` and passes it to `cat`. 一个是cat打开，一个是shell打开输入到cat
J	Ensure that you do not have a file named `foo` under the current directory. If you do, remove it first. Then try the following two command lines and think about why there is a difference between the outputs of them: `$ cat foo 2> error1` `$ cat < foo 2> error2` In the second command line, has the file `error2` been created? 第二个没创建，因为第二个是两个命令，第二个命令输出错误，但第一个命令没输出，错误后不执行第二个

1.2 Pipes

A pipe serves as an interconnection (set up by the shell) between 2 commands. It connects the output of a command to the input of another. For example,

```
$ who  |  wc -l
```

Here the pipe (represented by `|`) connects the output of `who` to the input of `wc` (`who` is said to be *piped* to `wc`). The above pipeline simply displays the number of users who are currently logged in. Without using a pipe, you would have to use two command lines for the purpose:

```
$ who > userlist
$ wc -l < userlist
```

Let's take a look at another example:

```
$ who | tee user.lst
```

Here the `tee` command saves the output of `who` in `user.lst`, and displays it on your screen as well. The output of `who` becomes the input of `tee`. The `tee` command reads from a standard input and writes to a standard output **and a file**. This allows the user to see an output, and in the meantime, save a copy of the output.

There is no restriction on the number of commands in a pipeline. The general structure is this:

```
command 1 | command 2 | … | command n
```

Some examples of longer pipelines are:

```
$ man who | grep "file" | lp
```
(In the man pages of `who`, retrieve all the lines which contain the keyword "`file`", and then print these lines)

```
$ cat /etc/passwd | grep "/bin/bash" | wc -l
```
(How many users have been set `bash` as their default shell?)

```
$ who | tee user.lst | wc -l
```
(Save the output of `who` in `user.lst`, and display the number of users)

A	Follow the exercises in section 1.1 to perform the following actions, and ensure that you understand the outputs.
B	`$ cat files` `$ cat files | sort` `$ cat files | sort | wc -l` `$ cat files | sort | wc -w` l 统计行。w统计字数
C	`$ who` `$ who | wc -l` `$ who | tee userlist` tee 读取输出到文件 `$ ls` `$ cat userlist` `$ who | tee userlist2 | wc -l` `$ ls` `$ cat userlist2`
D	What does the following command line do?

	`$ cat /etc/passwd | grep "David" | wc -l`
E	**Develop a single command line (using pipe) to display all the filenames stored under the `/bin` directory, and in the meantime, save a copy of the output into a file named `commands`. Try to display the filenames in multi-column format with the option `-x`. Verify that the file `commands` has been created with the right content. (5 marks)** ls -x /bin | tee commands

1.3 Two Special Files: `/dev/null` and `/dev/tty`

The file `/dev/null` accepts any stream without growing in size. It is a pseudo-device not associated with any physical device:

```
$ cat bar 2> /dev/null
```

The above command line redirects the error message (if the file `bar` does not exist) into the null device (a device which does not exist). Essentially this simply means getting rid of the error message.

When you use `find` from an ordinary user account to start its search from root, you get a lot of "Permission denied" error messages. You should now have a better understanding about the following command line (which you have previously used):

```
$ find / -name "*.c" -print 2>/dev/null
```

The file `/dev/tty` indicates your terminal:

```
$ who > /dev/tty
```

The above redirects the output of the `who` command to your terminal. One would argue that this is not necessary because you can just run the `who` command which simply displays its output on your terminal. This is true. But this special device file is useful when you want to refer to your terminal in some specific arrangement. For example,

```
$ who | tee /dev/tty | wc -l
```

This displays the list of users (without the need to save the list) and the number of users as well. Without this special device file, you would probably have to run 2 command lines to serve the same purposes.

Additionally, if you know somebody's terminal is, for example, `/dev/tty02`, you could send them a message like the following:

```
$ echo  Hello  >  /dev/tty02
```

The user working on this terminal could see it (if this feature is not disabled by the system administrator).

Which terminal are you using? Run the `tty` command to find out. (Warning: if this feature is enabled, do not send some message to disturb others while they are working hard!)

1.4 Command Substitution

The shell enables another way to connect two commands: Command Substitution.

In UNIX, The echo command is used for displaying messages. For example:

```
$ echo Hello World
Hello World  (the output)

$ echo "Monday Tuesday Wednesday"  (here the quotes are optional)
Monday Tuesday Wednesday  (the output)
```

Then how do we echo today's date with a message like the following:

today is Wed July 16 15:31:02 2016
(The underlined part must be the output of the `date` command)

The way to do this in UNIX is as follows:

```
$ echo today is `date`
today is Wed July 16 15:31:02 2016  (the output)
```

**Note the use of the ` (back-quote), which is not the single quote**. Here the shell executes the `date` command first, and then inserts its output into the message line, which is then displayed. This is called Command Substitution. Let's take a look at another two examples:

```
$ echo the current directory is `pwd`
the current directory is /u/bin  (a sample output)

$ echo there are `who | wc -l` users in the system
there are 37 users in the system  (a sample output)
```

You can use double quotes for Command Substitution:

```
$ echo "today is `date`"
today is Wed July 16 15:31:02 2016   (a sample output)

$ echo "the current directory is `pwd`"
the current directory is /u/bin   (a sample output)
```

However, it is a different story when single quotes are used:

```
$ echo 'today is `date`'
today is `date`   (the output)

$ echo 'the current directory is `pwd`'
the current directory is `pwd`    (the output)
```

Here the single quotes have removed the special meaning of the back-quotes for Command Substitution. Refer back to the previous introduction about Escaping on UNIX, which mentions that double or single quotes can be used for escaping on a UNIX shell. However, **here you see the first difference between the use of double quotes and single quotes in UNIX**:

Double quotes allow the use of Command Substitution, but single quotes do not.

A	Perform the following actions.
B	`$ echo This is a nice day` `$ echo "This is a nice day"` `$ echo 'This is a nice day'` (note the single quotes)
C	`$ echo This tutorial is held on `date`` (note the back quotes) `$ echo "This tutorial is held on `date`"` `$ echo 'This tutorial is held on `date`'`
D	`$ echo There are `ls | wc -l` files in this dir` `$ echo "There are `ls | wc -l` files in this dir"` `$ echo There are 'ls | wc -l' files in this dir` (note the single quotes) `$ echo There are \`ls | wc -l\` files in this dir`
E	**Make two subdirectories (under the current directory) named `dir1` and `dir2`, respectively. Explain the behaviour of the following command line to your tutor:** **`$ cd `find . -type d | tail -1``** **You may wish to explore the `tail` command first. (5 marks)**

(The End)

find . -type d => all folder under the directory
find . -type d | tail -1 => the last folder under the directory
cd `find . -type d | tail -1` => enter the last folder