CST8177 - Lab #4

Student Name:	Student Number:	Lab section:

Shell Features and File Globs

In-Lab Demo

1. Execute a command line using redirection
2. Demonstrate shell features, including at least variable expansion and command substitution
3. Demonstrate file globbing of several varieties

Working with the shell

In Unix and Linux, "shell" is the term used to describe the command-line interpreter. In this course, we will use the **`bash`** shell because of its wide popularity on Linux, although there are other shells available such as **`csh`** and **`sh`**.

Exercise #1 (review): Working with shell features

File name completion

Purpose	Command line example
Complete a file name (to avoid typing long file names or making a typo in a complex name)	`cp .bashrc abc.bashcopy` `vi abc[TAB]`

Aliasing

Purpose	Command line example
List all aliases	`alias`
Create an alias	`alias ll="ls -l"`
Remove an alias	`unalias ll`

History Mechanism

Purpose	Command line example
Display the current history buffer	`history | less`
Re-execute the last command	`!!`
Re-execute any previous command using keyboard keys	Use the up and down arrow keys
Re-execute any previous command using the event number	`!`***`n`*** (where ***`n`*** is the event number as listed in the **`history`** output)
Edit a command	Use the up and down arrow keys to select the command and edit it

Redirection & command grouping

A file descriptor is a numeric identifier (small unsigned integer) that a UNIX system uses to identify an open file that is attached to a process. *Note*: A file descriptor is created by a process through issuing an **`open()`** system call for the file name. A file descriptor ceases to exist when it is no longer held by any process. By default a process is set up with three file descriptors:

- **`0`** (**`stdin`**), usually associated with the current console (**`/dev/pts/`*`n`***) - keyboard
- **`1`** (**`stdout`**), usually associated with the current console (**`/dev/pts/`*`n`***) - screen
- **`2`** (**`stderr`**), usually associated with the current console (**`/dev/pts/`*`n`***) - screen

Note that **`pts`** is used for a terminal window, a pseudo-terminal. An actual terminal will be **`tty`**.

Create an empty directory and **`touch x`** to create a file in it. Use the command **`ls x y`** to provide examples of the actions listed below. Use only one command for each.

Purpose	**Command line solution**
Redirect **`stdout`** to a file named **`out`** and view the file.	
Redirect **`stderr`** to a file named **`err`** and view the file.	
Redirect the output such that **`stdout`** is written to **`out`** and **`stderr`** is written to **`err`** from the single command.	
Append **`stdout`** to a file named **`out`** and view the file.	
Append **`stderr`** to a file named **`err`** and view the file.	
Redirect **`stdout`** to a file named **`out`** and redirect **`stderr`** so it is written to the same file as **`stdout`** (but do not name the file for **`stderr`**). View the file **`out`**.	
Redirect both **`stderr`** and **`stdout`** to a file named **`out`** and view the file.	
Reverse the redirection sequence in the question above and repeat. What's the difference, if any?	

There are also some useful command separators you should be aware of. Try each of these (to cause **mkdir dir** to fail, create it twice in the same directory):

- **;** - the semi-colon executes one command at a time and is equivalent to the **[Enter]** key between commands.
  Example: **mkdir dir; cd dir**
- **&&** - the double ampersand executes the second command only if the first command executed successfully.
  Example: **mkdir dir && echo Directory creation successful**
- **||** - the double pipe executes the second command only if the first command executed **un**successfully.
  Example: **mkdir dir || echo Could not make new directory**

Exercise #2: Working with shell expansion/substitution

Execute the commands listed below, record the output here, and examine the output.

Brace expansion

- **echo post{script,office,ure}**
- **mkdir -p /home/teacher/{CST8207,CST8177}/{F,W}0{0,1}**

How many directories have been created (excluding **teacher**)? _____

Tip: Use **find /home/teacher | wc -l** and adjust as needed.

Tilde expansion

- **ls -d ~** _____

Variable expansion

- **echo "My search path is: $PATH"** _____

  _____

- **cd /**
- **ls -d "home"** _____
- **ls -d "HOME"** _____
- **ls -d "$HOME"** _____
- **ls -d "${HOME}"** _____

Command substitution

- **ls -ld $(find /home -maxdepth 1 -uid 500)**

  _____
- Compare with: **find /home -maxdepth 1 -uid 500 -ls**

  _____

Briefly describe the significant differences _____

_____

Arithmetic Expressions

- **`echo Result $(( 3 + 6 - 2 ))`** _____
- **`echo Result $(( 67 + 3 / 2 ))`** _____
- **`echo Result $(( (8 - 4) * 2 ))`** _____
- **`echo Result $(( 8 - 4 * 2 ))`** _____
- **`echo Result $(( 5 + 4 ))`** _____

File globbing

- Create a directory called **`dir`**.
- Create files in the current directory: **`touch x{1,2,3} x{1,2,3}0`**
- **`cp x?0 dir`**

  How many files have been copied (see above for counting tips)? _____
- Delete all files in the current directory (do not use **`-f`** or **`-r`**): **`rm x*`**
- Delete the directory **`dir`**
- **`touch hda hdb hdc hdd`**
- **`ls hd[abc]`** What is the output of this command?

  _____
- **`ls hd[a-c]`** What is the output of this command?

  _____
- **`rm hd[a-d]`**
- **`ls hd[a-d]`** What is the output of this command?

  _____
- **`touch hda1 hdb2 hdc3 hdd4`**
- **`ls hd[b-d][1-3]`** What is the output of this command?

  _____
- **`rm hd[a-d][1-4]`**
- **`rm hd[a-d][1-4]`** What is the output of this command?

  _____

Exercise #3: Working with metacharacters

- The special character: question mark, escaped by the backslash
  - **`touch your        # Create this empty file`**
  - **`echo How are you?`**

- **`echo How are you\?`** What is the difference between these commands?

- The special character: single quote, escaped by the backslash
  - **`echo Don't you need $5.00?`**

    Note: Use **`Ctrl+D`** to terminate the input prompt **`>`**
  - **`echo Don\'t you need $5.00?`**
  - **`echo Don\'t you need '$5.00?'`**
  - Write your own **`echo`** command, using another method to allow the **`$`** and **`?`** to print correctly:

- The special character: double quote, escaped by the single quote
  - **`echo Mother yelled "Time to eat!"`**
  - **`echo 'Mother yelled "Time to eat!"'`**
  - Write your own **`echo`** command, using another method to allow the **`"`**s to print correctly:

Exercise #4: Working with quotes

Remove the special meaning that the space, the newline, and the **`#`** have for the shell.

- This command does not produce the desired result: **`useradd -c Alan Turing enigma`**
- Each of these will work fine (delete all but one with **`userdel`**)
  - **`useradd -c "Alan Turing" enigma`**
  - **`useradd -c 'Alan Turing' enigma`**
  - **`useradd -c Alan\ Turing enigma`**
- This command does not produce the desired result: **`grep Alan Turing /etc/passwd`**
  - Fix it using double quotes:

  - Fix it using single quotes:

  - Fix it using the escape character:

- This command does not produce the desired result: **`alias la=ls -A`**
    - Fix the command line using double quotes:

- To execute a statement that spans several lines, use the escape character (**`\`**) immediately before **`[Enter]`**. Show the result of the line-spanning **`echo`**:
    - **`echo hello sailor[Enter]`**
    - **`echo hello \[Enter]`**
      `sailor[Enter]`

- Test a line-spanning **`echo,`** but put a space after the **`\`** and before the **`[ENTER]`**. What happens?

- Show a correctly working multi-line **`grep`** or **`alias`** command:
