Outline

- Introduction - history
- Command line basics – getting help
- File system
- Working with files and directories
- More file handling
- ➢The shell revisited
- Monitoring resources

Outline details

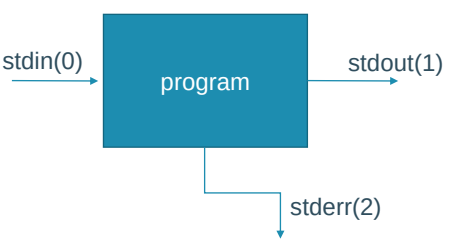
- I/O redirection
- Pipes
- Command chaining
- Very basic scripting
- Some (useful) processing commands

IO redirection

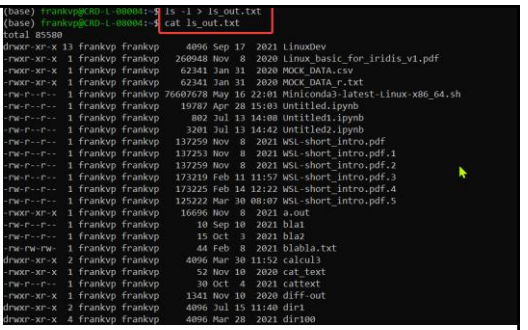
Input / output

- Inputs and outputs of a program are called streams in Linux.
- **stdin** (**st**andar**d in**put) – stream data going into a program. By default, this is input from the keyboard.
- **stdout** (**st**andar**d out**put) – output stream where data is written out by a program. By default, this output is sent to the screen.
- **stderr** (**st**andar**d err**or) – another output stream (independent of stdout) where programs output error messages. By default, error output is sent to the screen.

https://cvw.cac.cornell.edu/Linux/io



Redirection



- Redirection is all about files
- Output redirect out from screen to a file
  - This can be done with the redirection operator `>`
    `$ ls -l > ls_out.txt`
  - Redirection will create the named file if it doesn't exist, or else overwrite the existing file.
- Append with the operator `>>`
  - append instead of rewriting the file
    `$ ls -l >> ls_out.txt`
- Input redirection
  - Input can also be given to a command from a file instead of typing it in the shell by using the redirection operator `<`
    `$ sort < tabel.dat`

Redirection

- Error Redirection
  - Normal or standard output (stdout), will not affect stderr (separate stream).
  - Use the redirection operator `2>`
    `$ ls non-existing* 2> ls_err.txt`
- Trash any data
  - `/dev/null` is a special file that is used to trash any data that is redirected to it. Any output that is sent to /dev/null is discarded.
    `$ ls > /dev/null`
- See also: https://www.tecmint.com/linux-io-input-output-redirection-operators/

Pipes

Pipes

- Piping is all about processes
- Pipes (also referred to as pipelines) can be used to direct the output of one command to the input of another.
  The Shell arranges it so that the standard output of one command is fed to another command
- use the **|** key on the keyboard
- `$ ls -l | less`
  - the output of the ls command is piped into the less program
  - compare with `$ ls -l > less` (the output of the ls command is saved in a file with the name less)

pipes

- Key feature of the unix philosophy (https://en.wikipedia.org/wiki/Unix_philosophy)
  - Write programs that do one thing and do it well.
  - Write programs to work together.
  - Write programs to handle text streams, because that is a universal interface.

Redirection revisited: tee

- Show and Save Output
- When redirecting output to a file, nothing is shown on the screen.
- To have the output go to both a file and the screen, use the `tee` command
  - Syntax, use it with a pipe

```
[command] | tee [options] [filename]
$ ls  -al | tee ls_out.txt
```

  - Option `-a`: append (instead of overwriting)

```
$ ls  -al | tee -a ls_out.txt
```

Command chaining

Command chaining

- ; makes it possible to run, several commands in a single go and the execution of command occurs sequentially.
- && (AND Operator) will execute the second command only, if the execution of first command SUCCEEDS, i.e., the exit status of the first command is 0.
- ||(OR Operator) will execute the second command only if the execution of first command fails, i.e., the exit status of first command is '1'.

Command	What does it?
cmd1 ; cmd2	Run *cmd1* then *cmd2*
cmd1 && cmd2	Run *cmd2* if *cmd1* is successful
cmd1 || cmd2	Run *cmd2* if *cmd1* is not successful
cmd &	Run *cmd* in a subshell

Command substitution

- Use `$( )` to insert the output of one command into the arguments for another command
- Example:

```
which ls
# the classic way of working
cp -v /usr/bin/ls .
# a smarter way ?!
cp -v $(which ls) .
```

(very) Basic shell scripting

Basic steps

1. **Create a file using** an editor(Nano). Name script file with extension **.sh**
2. **Start** the script with **#! /bin/bash**
3. Write some code.
4. Save the script file
5. For **executing** the script type: **`bash filename.sh`**

variables

- define a variable by using the syntax
  `variable_name=value`
- To get the value of the variable, put $ before the variable.
- File: script_hello_world.sh

```
#!/usr/bin/bash
# a first example using variables
greet='hello'
name='world'
echo $greet $name
```

Linux variables

- Variables are a way of passing information from the shell to programs
- 2 categories:
  - shell variables.
    - shell variables apply only to the current instance of the shell and are used to set short-term working conditions;
  - environment variables
    - environment variables reach further, and those are set at login are valid for the duration of the session.

https://projects.ncsu.edu/hpc/Documents/unixtut/unix8.html

https://linuxize.com/post/how-to-set-and-list-environment-variables-in-linux/

environment

- The shell environment is configured through a variety of variables called *environment variables*. This environment tells the shell and your various tools how to behave.
- `env`: prints out the environment in its current state, listing all environment variables.
- SHELL, HOME, and PATH are built-in shell variables.
- Any variable can be declared to the shell by typing
  `MYVAR=something`
  - Convention: declare shell variable in capitals.
  - The variable can then be used in bash commands or scripts by inserting $MYVAR. If you want to insert the variable inside a string, this can be done by ${MYVAR} with the beginning and ending portions of the string on either side. For example:

```
$ DATAPATH=/home/jolo/Project
$ ls $DATAPATH
$ cp newdatafile.txt ${DATAPATH}/todaysdatafile.txt
$ ls $DATAPATH
```

https://cvw.cac.cornell.edu/Linux/envvars

https://linuxize.com/post/how-to-set-and-list-environment-variables-in-linux/

environment

- PATH stores a list of directory paths which the shell searches when you issue a command.
- view this list with `echo $PATH`.
- If the command you issue is not found in any of the listed paths (having been added either by the OS, a system administrator, or you), then the shell will not be able to execute it, since it is not found in any of the directories in the PATH environment variable. You can change the directories in the list using the export command. To add directories to your path, you can use:
- `$ export PATH=$PATH:/path/to/new/command`
  - The `:` joins directories in the path variable together.
- If you wanted to completely replace the list with a different path:
  - `$ export PATH=/path/to/replacement/directory`
  - Please keep in mind that the previous list will be erased.
- https://cvw.cac.cornell.edu/Linux/envvars

Processing commands

sort

Command	Options	What does it?
sort		Sort anyfile lexicographically
	-n	Sort in numeric order
	-r	sort things in reverse order
	-R	Re-arrange the lines of the input randomly

- sort is the tool to sort a file. Sorting by itself isn't that useful, but it is an important pre-requisite to a lot of other tasks.
- Note: -R can be useful for developing a large number of test cases.
- File: sort_demo.sh

- See also: https://shapeshed.com/unix-sort/

tr

Command	Options	Common usage	What does it?
tr		tr SET1 [SET2]	it will replace each character in SET1 with each character in the same position in SET2. When SET2 is shorter than SET1, the tr command will, by default, repeat the last character of SET2
	-t		truncate SET1 to the length of SET2:
	-s		Squeeze, remove repeated instances of a character
	-d	tr -d SET1	delete characters in SET1.
	-c	tr -c SET1	search for a complement of SET1. i.e. searching for the inverse of SET1.

Translate, a tool that can:

- convert character case
- squeeze repeating characters
- delete specific characters
- do basic text replacement
- File: tr_demo.sh

wc

Command	Options	Common usage	What does it?
wc		wc anyfile	print newline, word, and byte counts for anyfile
	-c		print byte counts
	-m		print character counts
	-w		print word counts
	-l		print newline counts

- A tool that to obtain word counts and line counts.
- Line counts are usually a measure for counting a number of elements.
- File: wc_demo.sh

uniq

Command	Options	Common usage	What does it?
uniq		uniq anyfile	the default behaviour is to discard duplicate lines
	-c		count number of occurences
	-d		only print duplicate lines
	-u		only print unique lines
	-i		ignore case

- Tool to detect the adjacent duplicate lines and also deletes the duplicate lines.
- Note: uniq is not able to detect the duplicate lines unless they are adjacent to each other, therefor it is often seen in a pipe coupled with sort.
  Or simply use sort -u instead of uniq command
- File: uniq_demo.sh

comm

Command	Options	Common usage	What does it?
comm		comm file1 file2	compare two (sorted) files line by line column 1: only in file1 column 2: only in file 2 column 3: in file1 and file2
	-1		suppress column 1
	-2		suppress column 2
	-3		suppress column 3
	--total		Output a summary

- comm compares each and every line of the files and displays the unique lines and common lines of the files in separate columns
- File: comm_demo.sh

Slicing and glueing files

- Slice rowlike (horizontal)
  - split
  - cat
- Slice columnlike (vertical)
  - cut
  - paste

split

Command	Options	Common usage	What does it?
split		split filename prefix	split filename and prefix is the name you wish to give the small output files
	-b		the number of bytes in each of the smaller files.
	-l		the number of lines of the smaller files (default is 1,000).
	-n		the number of parts to split the original file

- Split a file into smaller chunks
- Specify in what way the file needs to be cut
- File: split_demo.sh

cat

Command	Options	Common usage	What does it?
cat		cat file1 file2 filen	display the content of multiple files
	-s		Omit the empty lines
	-n		display the line number of file content

- Concatenate files together into a larger file
- File: split_cat_demo.sh

cut

Command	Options	Common usage	What does it?
cut			cut option filename
	-d	cut -d ',' -f 1 filename	specify a delimiter to use instead of the default TAB delimiter
	-f	cut -f n cut -f n1-n2 cut -f n1,n2,n3	a specified field, a field set, or a field range
	-b	cut -b n cut -b n1-n2 cut -b n1,n2,n3	cut by byte position cut by byte position range cut by byte position list
	-c	cut -c n cut -c n1-n2 cut -c n1,n2,n3	Similar to b, but by character position, some characters take more than a byte for the representation
	--output-delimiter	cut -c 2-5,8 —output-delimiter=@	Add a custom delimiter for output

- extracts a column of columns of information from a file (usually delimited)
- File: cut_demo.sh

paste

Command	Options	Common usage	What does it?
paste			
	-d	paste -d ',' file1 file2 filen	specify a delimiter to use instead of the default TAB delimiter. A list of delimiters can be specified
	-s	paste -s file1 file2 filen	It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab.

- combine files as columns into 1 file
- File: paste_demo.sh

wget

- Instead of downloading files from your browser, just copy and paste their URL and download them with `wget`
- main features
  - http and ftp support
  - Can resume interrupted downloads
  - Can download entire sites or at least check for bad links
  - Very useful in scripts or when no graphics are available (system administration, embedded systems)
- `$ wget -v https://github.com/franklbvp/linuxintro/blob/master/docs/WSL-short_intro.pdf`
- https://bioinformaticsworkbook.org/dataAcquisition/fileTransfer/downloading-files-via-wget.html#gsc.tab=0

The grep command

- **G**lobal **r**egular **e**xpression **p**rint
- Grep is used to search text files with **regular expressions** (**regex**).
  - It prints the lines matching the given pattern in a text file.
  - If no file is given, grep will recursively search the given pattern in the files in current directory
  - regex is different from wildcards!

grep

Command	Options	Common usage	What does it?
grep		`grep pattern file`	Prints all lines in file matching a particular search pattern
	-i	`grep -i pattern files`	Case insensitive search
	-r	`grep -r pattern path`	Recursive search through the subdirectories
	-v	`grep -v pattern files`	Inverted search, print all lines NOT containing the specified pattern
	-o	`grep -o`	Show matched part of file only
	-n	`grep -n pattern file`	Include line numbers
	-w	`grep -w pattern files`	only returns lines where the sought-for string is a whole word and not part of a larger word.
	-L	`grep -L pattern files`	outputs the names of files that do NOT contain matches for the search pattern
	-l	`grep -l pattern files`	outputs the names of files that do contain matches for the search pattern

Regular expressions for grep command

Expression	Description	Example
`*`	Any number of the preceding character is allowed	`12*3 matches "13", "123", "1223", "12223"`
`.`	Matches any single character (except newlines, normally)	`c.t matches "cat", "cut" or "cot"`
`?`	The preceding character may or may not be present	`ma?ke matches "make", "mke"`
`^`	Appears at beginning	`^he matches "hello", "hell", "help", "he is a boy"`
`$`	Appears at end	`^he matches "hello", "hell", "help", "he is a boy"`
`[nnn]`	Match any character in this set. - defines ranges (e.g. [a-z] is any lowercase letter), ^ means "not"	`[abc] matches "a","b" or "c" in the string "abc". a[^b]c matches "aec", "acc", "adc", but not "abc"`

The grep command

- `$ grep <pattern> <files>`
  Scans the given files and displays the lines which match the given pattern.
- `$ grep error *.log`
  Displays all the lines containing error in the *.log files
- `$ grep -i error *.log`
  Same, but case insensitive
- `$ grep -ri error .`
  Same, but recursively in all the files in the current directory and its subdirectories
- `$ grep -v info *.log`
  Outputs all the lines in the files except those containing info.
- http://www.thegeekstuff.com/2009/03/15-practical-unix-grep-command-examples/

More commands

- `$ sleep 60`
  Waits for 60 seconds
  (doesn't consume system resources).
- `$ date`
  Returns the current date. Useful in scripts to record when commands started or completed.

time

- Helpful command for doing simple benchmarking
- Run your scripts along with **time** command, and compare the execution time.
- Example:
  ```
  $ time ls
  ```
  real 0m2.304s (actual elapsed time)
  user 0m0.449s (CPU time running program code)
  sys 0m0.106s (CPU time running system calls)
- Timing (https://stackoverflow.com/questions/556405/what-do-real-user-and-sys-mean-in-the-output-of-time1/556411#556411):
  - **Real**: wall clock time - time from start to finish of the call.
  - **User** is the amount of CPU time spent in user-mode code (outside the kernel) *within* the process. This is only actual CPU time used in executing the process. Other processes and time the process spends blocked do not count.
  - **Sys** is the amount of CPU time spent in the kernel within the process. This means executing CPU time spent in system calls *within the kernel,* as opposed to library code, which is still running in user-space.