REDIRECTION

- Direct the flow from one place to another
- In Linux, one can redirect the flow of input or output from their normal default location.
- Redirect output to save the result of a command.
- Input redirection, the default location for a command's input is changed.
  * Used less often than output redirection
  * Most commands that accept filenames as part of their syntax usually don't require the input redirection operator.

LINUX REFERS TO

- STANDARD INPUT (stdin) as default location for inputting commands. (Keyboard – default)

- STANDARD OUTPUT (stdout) as default location for outputting commands (Screen – default)

- STANDARD ERROR (stderr) as default location of errors generated from commands. (screen – default)

REDIRECTION ALLOWS YOU TO CHANGE THE DEFAULT LOCATIONS OF STDIN, STDOUT, and STD ERR.

FILE DESCRIPTORS

FILE	DESCRIPTOR	LOCATION
Standard input	0	/dev/stdin
Standard output	1	/dev/stdout
Standard error	2	/dev/stderr

Linux uses file descriptors to handle input, output, and error handling.

Redirecting Standard Input

command < filename ⇐ Causes the command to read from a file instead of the keyboard.

Example, sort command typed in without any options sorts the data from stdin. (keyboard).

You can sort a file if you use a redirection symbol, sort < personnel.txt

Redirecting Standard Output

- Useful when you want to keep the output of a command for later review.

command > filename

If filename exists, then contents are overwritten

If filename doesn't exist, then file is created and results are written into the file.

Example, ls > lis list.txt

who > file2.txt

You can redirect input and output in a single command.

Example, sort -r < input.txt > output.txt

sorts the file input.txt in reverse order, and then redirects that output to output.txt.

Redirecting Output to Append to a file:

You use the redirect append (>>) Operation to preserve the contents of a file.

command >> filename

If file exists, contents are appended to end of file

If file doesn't exist, file is created.

Redirecting Standard Error

— Most commands send their error messages to the screen.

— Error can be captured for later review.

— Redirecting similar to Standard Output, but must refer to file descriptor 2 for proper redirection

Example 1. `rm 2> rmlist.err`

Error messages from rm command redirected to rmlist.err.

2. Typically, Standard error is combined with standard output.

To redirect to two different files:

`rm > rmlist.txt 2> rmlist.err`

3. To redirect, standard error and standard output to same file use 2>

`rm 2> rmlisting.txt`

Example,

1. (ls -l ; who ; rm t5) > listing.txt
   2> errlisting.txt

Provides long listing of files in directory followed by users who are logged in followed by removing a file called t5 (if it exists)

Output goes to → listing.txt

If t5 is not removed error goes to → errlisting.txt

2. (ls -l ; who ; rm t5) &> combolisting.txt

Same as before

Error and output goes to combolisting.txt

PATTERN MATCHING

- Technique that uses metacharacters to match characters based upon a certain pattern.
- Useful for searching specific files or directories
- Used in data mining and data processing tasks. (REGULAR EXPRESSIONS)

* (wildcard) — Match any character

? — Match a single character

[....] — Match one or several characters.

↑

.... refers to the specific characters you want to match.

Matching any Character (*)

- Use the asterisk symbol to match any character.
- This type of pattern matching usually concludes in the largest number of results returned because it matches any character.

Examples,

1. cat *.dat /dirA
   ↳ Copy all files with a "dat" extension to the directory dirA.
2. ls -l t*
   ↳ List all files begining with lowercase "t".
3. rm G*.txt
   ↳ Remove all files starting with uppercase "G" and extension "txt"

Matching a single character ?

- To match a single character position, you use the ? symbol.

- Pattern matching is more restrictive because you can control which character position to match.

Examples, 1. ls ???2*

↳ list all files where 2 is in the 4th character position in the filename.

If you do, ls *2*

↳ list files with 2 anywhere in the filename.

So, [?] is required to establish proper placement of the character for which you are searching.

2. ls t?.dat

↳ list files that begin with "t" followed by a single characters and has a "dat" extension.

3. ls ????.???

↳ list files containing four characters with a 3-characters extension.

4. rm t?.???

↳ remove files begining with t having only one remaining characters in the 2nd position and having a 3-characters extension.

Match any one of Several characters [ ]

— ([ ]) Square brackets allow you to match any one of the characters contained within the square brackets.

— Allows you to match a range of characters.

Examples 1. ls [L-P]*

↳ lists files that begin with "L" through "P".

2. ls [ft]*

↳ lists files that have filenames starting with "f" or "t"

3. ls tel[le]* ← tele@commute
← tel@.dat

↳ lists files beginning with "tel" and having either an "l" or "e" in the fourth character position.

Understanding the Use of Quoting

- If you want to use any of the metacharacters, you need to distinguish between the literal symbol and the symbol's use as a metacharacter.

↑ THIS IS CALLED QUOTING

Characters Used for Quoting:

- Backslash (\) ← Escape character
- Single quote (')
- Double quote (")

Escape character

Examples 1) If you want to create a file with the ">" in the filename, then

touch <test> → will result in error because shell interpret < and >

touch \<test\> → you escape the characters.

2) - Single Quote also protects the literal meaning of metacharacters.

- Differs from backslash (\) in that it protects all characters within the single quotes.

Single Quotes

(a) echo 'Joe said "Have fun!"'
↳ results in placing double quotes around "Have fun!"

(b) A single quote cannot occur with other single quotes even if preceded by a backslash.

echo 'Joe said 'Have Fun'' → would not result in single quotes around Have Fun.

DOUBLE QUOTES

3) Double Quoks

- Used to protect all symbols and characters wiltten double quotes.
- It will not protect: $, ', \

(a) echo '$5.00' but not echo "$5.00"

(b) ~~echo~~ touch "Mike's File"
↳ creates the file with filename Mike's File

when using single quote as an apostrophe

UNDERSTANDING EXPANSION

- Expansion is the process of changing metacharacter and special symbols into something else.

- The Shell uses symbols to expand or substitute words that are entered on the command line.
  e.g. $HOME expands to home directory

Tilde Expansion

~ → expands to user's home directory

cd ~ → change to home directory

echo ~+ → substitutes for current working directory or PWD

echo ~- → Substitutes for OLDPWD.

${name:offset:length}

↑

Used for specifying a portion of a parameter's value.

name → name of variable

offset → begining position (begins at 0)

length → number of positions of the value you want.

example, p="56789"

echo ${p:0:2}

↳ displays 56

echo ${p:4:1}

↳ displays 9

echo ${#p}

↳ display number of character in the variable (or the length)

COMMAND SUBSTITUTON

- Set a variable to equal the output of a command.
- Then display the contents of the variable using parameter Substitution.

$(command)

Example, 1. t=$(ls)

echo $t → Display file listing

2. t=$(ls $(pwd))

echo $t → Display file listing of current directory

ADDITIONAL COMMANDS

find files

(A) **find** → - allows you to search for files in the directory tree
- Displays full path to the file

examples,

- `find -mtime 5`
  ↳ finds files modified 5 day ago.
- `find -name .html`
  ↳ find file with a specific name called "html"
- `find -size 1k`
  ↳ find files that are 1kb or larger
  ← b (files)
- `find -type d`
  ↳ find files of certain types, here looking for directories.
- `find -user username`
  ↳ find files owned by the username.

(B) tr → translates or deletes characters.

-d option
↳ to delete characters

tr option set1 set2

↳ characters in set 1 are translated to characters in set 2.

- tr [:lower:] [:upper:]
  ↳ characters are translated to uppercase
- tr y Y
  ↳ translates lowercase y to uppercase Y.

tr → translate

(C) tee → Reads from standard input and writes to both standard output and to files.

tee option filename

- who | tee output.txt
  ↳ User logged in displayed on screen and saved in the file
- who | tee -a output.txt
  ↳ appends to the output.txt file

tee