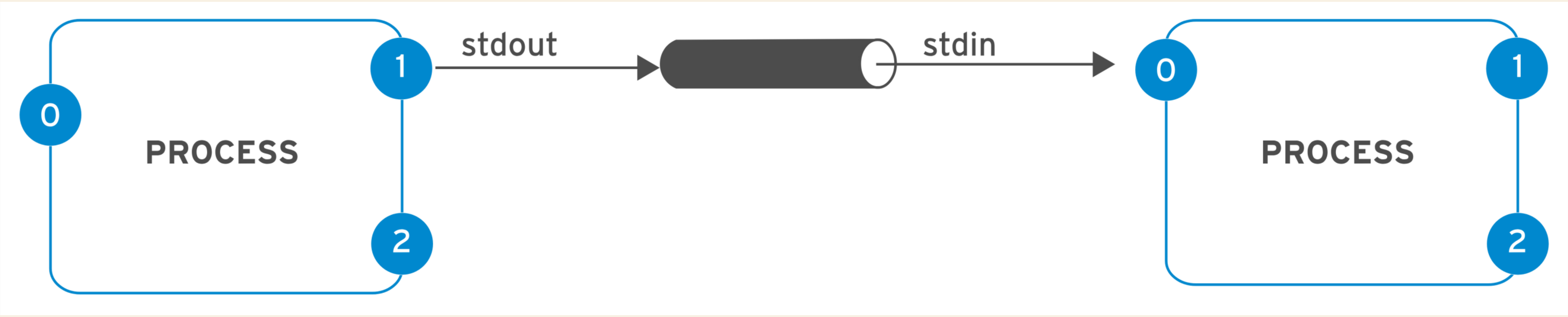
DAY 5

Construct Pipelines

A pipeline is a sequence of one or more commands that are separated by the vertical bar character (|).

A pipeline connects the standard output of the first command to the standard input of the next command.



Pipelines and I/O redirection both manipulate standard output and standard input. Pipelines send the standard output from one process to the standard input of another process. Redirection sends standard output to files or gets standard input from files.

```
ls -l /usr/bin | less
```

```
ls | wc -l
```

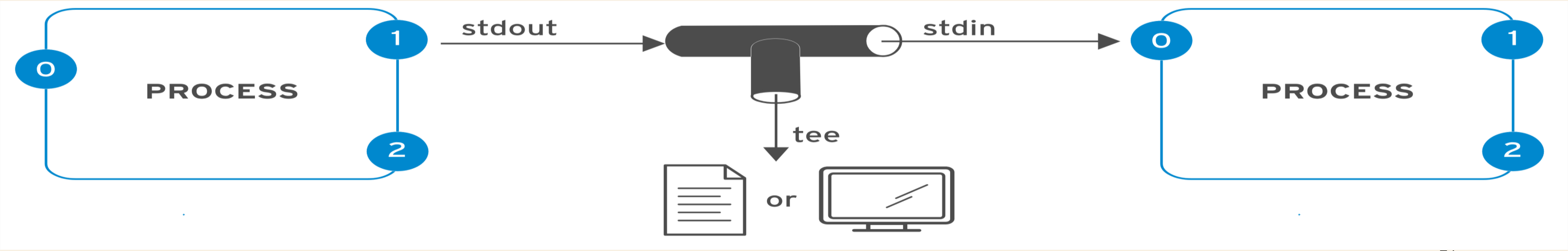
Pipelines, Redirection, and Appending to a File

When we combine redirection with a pipeline, the shell sets up the entire pipeline first, and then it redirects the input/output. If you use output redirection in the middle of a pipeline, then the output goes to the file and not to the next command in the pipeline.

In the next example, the output of the ls command goes to the /tmp/saved-output file, and the less command displays nothing on the terminal.

```
[user@host ~]$ ls > /tmp/saved-output | less
```

The tee command overcomes this limitation. In a pipeline, tee copies its standard input to its standard output and also redirects its standard output to the files that are given as arguments to the command. If you imagine data as water that flows through a pipeline, then you can visualize tee as a "T" joint in the pipe that directs output in two directions.



The next example redirects the output of the ls command to the /tmp/saved-output file and passes it to the less command, so it is displayed on the terminal one screen at a time.

```
[user@host ~]$ ls -l | tee /tmp/saved-output | less
```

If you use the tee command at the end of a pipeline, then the terminal shows the output of the commands in the pipeline and saves it to a file at the same time.

```
[user@host ~]$ ls -t | head -n 10 | tee /tmp/ten-last-changed-files
```

Use the tee command -a option to append the content to a file instead of overwriting it.

```
[user@host ~]$ ls -l | tee -a /tmp/append-files
```

We can redirect standard error through a pipeline, but you cannot use the merging redirection operators (&> and &>>). The following example is the correct way to redirect both standard output and standard error through a pipeline:

```
[user@host ~]$ find / -name passwd 2>&1 | less
```

find command

find command in Unix is used to find a file/directory in the filesystem hierarchy and perform operations on them. It allows you to search based on:

- ✓ File name
- ✓ Type of the file
- ✓ Permission
- ✓ Owner of the file
- ✓ Size
- ✓ No. of links etc.,

The basic syntax of a find command is :

find <search location> <criteria> <value for criteria>

find /root -name test
find /root -type f → files
find /root -type d → folders
find /root -type l → links

Find based on Size:

i) To find a file having size of 10 bytes

```
find /root -type f -size 10c
```

ii) To find a file having size lesser than 10 bytes

```
find /root -type f -size -10c
```

iii) To find a file having size greater than 10 bytes

```
find /root -type f -size +10c
```

File Permissions:

A file in Linux has 3 permissions: read, write and execute. Each of them has an absolute value to them.

Read(r) has 4, write(w) has 2 and execute(x) has 1.
For a given file, there will be an owner, group owner and other users.
Each of these users will have specific permission on the file.
If all these users have full permission on the file, it means the permission of the file is 777(7 for owner, 7 for group owner, 7 for other users).
7 is the sum of total of the absolute values of all three permissions (read, write and execute).

To find a file based on the permission of the file, use the **-perm** option.

The value for -perm can be specified in 2 ways.

i) absolute way (777)
ii) symbolic way(u=rwx,g=rwx,o=rwx)

```
find / -type f -perm 755
find / -type d -perm u=rwx
```

Text processing commands:

Some of the common text processing tasks are:

1. **Displaying the content of a file efficiently**
2. **Search for a particular pattern in a file**
3. **Arranging the data in a file in order**
4. **Generating report from the data in a file**

# cat command is used to display/dump the content of a file on the terminal. **cat /etc/passwd**

# paste command is used to merge the lines of two or more files. **paste /etc/passwd /etc/shadow**

In the output we can see that, the first line of /etc/passwd and the first line of /etc/shadow is printed in a line separated by a tab space.

# Most of the times, administrators will be dealing with configuration files that has even more than a hundred lines. Using cat command to display such files is not much helpful since it just dumps the content on the terminal and not help us with moving around the file with ease. In this case, one can take the help of more/less command to display the content of large files in a more elegant way.

more /etc/sudoers **less /etc/sudoers**

The main differences is that the less command allows forward as well as backward movement of files whereas the more command allows only forward movement.

Filtering rows in a file:

head command is used to print the first part of a files. By default, it prints the first 10 lines of a file.

```
head /etc/passwd
head -n 6 /etc/passwd
```

tail command does the opposite of what head command does. It displays the last 10 lines of a file by default.

```
tail /etc/passwd
tail -n 5 /etc/passwd
```

Filtering columns in a file:

cut command is used to satisfy the above requirement. It prints only the section of each line of a file.

```
cut -d ":" -f1 /etc/passwd          cut -d ":" -f1,2,7 /etc/passwd          date | cut -d " " -f2
```

-d => delimiter i.e., the character in the file that is dividing one section of a line from other. In /etc/passwd, the delimiter is ":“

-f => fields i.e., which all sections/fields of the file need to be printed. The field number starts from 1.

sort command in Unix is used to sort text files or output. By default, it sorts text as dictionary sort and numbers in ascending order. By using the appropriate option, it can also be sorted in reverse order of dictionary and in descending order of numbers.

```
sort /users.txt

sort -r /users.txt  # reverse order

sort -n numbers.txt # numerical data

sort -n -r numbers.txt
```

<u>Sort a file based on a field:</u>

```
tail /etc/passwd | sort -nt ":" -k3
```

-n => since the third column contains numerical data.
-t => specifies the field separator. Here, the field separator is ":". The field separator must be mentioned immediately after the -t option. Otherwise, it will result in error.
-k => key, the column number based on which the file should be sorted.

To remove duplicate entries

```
uniq logon.txt
```

```
sort logon.txt  | uniq
```

To count the repeating entries

```
sort logon.txt | uniq -c
```

To list non-repeating entries

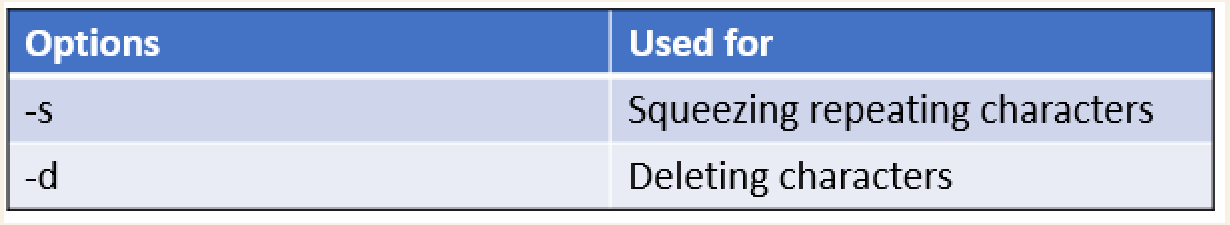
```
cat logon.txt | uniq –u
```

To list the duplicate entry

```
cat logon.txt | uniq –d
```

Translating, squeezing and deleting characters

“tr” command is a Linux utility used for translating, squeezing or deleting characters. tr command accepts input in the form of a file or through pipes. It can translate characters, squeeze repeating characters and delete specific characters in the given input.



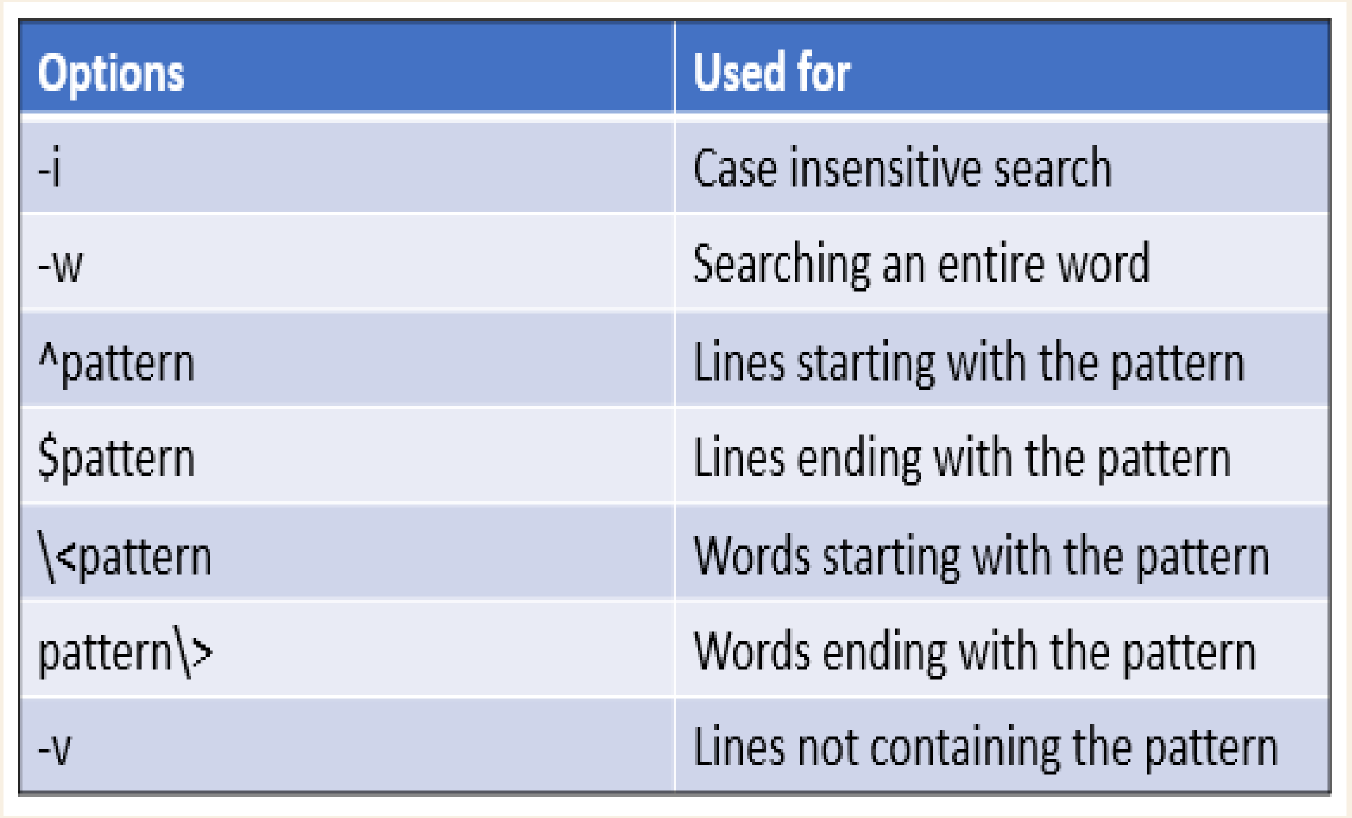
Options	Used for
-s	Squeezing repeating characters
-d	Deleting characters

Translate all lower case into upper case: `cat test.txt | tr [:lower:] [:upper:]`
Translate all upper case into lower case: `cat test.txt | tr [:upper:] [:lower:]`
To translate a character into another: `cat test.txt | tr 'E' ‘e’`
To squeeze repeating characters: `cat test.txt | tr -s ' ’`
To squeeze and translate: `cat test.txt | tr -s ' ' ':’`
To delete a character: `cat test.txt | tr -d 'i'`
To delete multiple characters: `cat test.txt | tr -d 'ie'`

PATTERN MATCHING

"**grep**" utility in Linux is used to match a given pattern in the file and print only those lines that contain the pattern.

Match the exact pattern:
```
grep “kumar" sample.txt
```
Match pattern without case sensitivity:
```
grep -i “kumar" sample.txt
```
Match lines starting with a pattern:
```
grep "^ku" sample.txt
```
Matching lines ending with a pattern:
```
grep '\.$’ sample.txt
```
Match an entire word:
```
grep -w “kumar" sample.txt
```
Match words starting with a pattern:
```
grep -i "\<kum" sample.txt
```
Match words ending with a pattern:
```
grep -i “s\>" sample.txt
```
Match all empty lines:
```
grep '^$’ sample.txt
```
Match all lines with content:
```
grep '.’ sample.txt
```



Options	Used for
-i	Case insensitive search
-w	Searching an entire word
^pattern	Lines starting with the pattern
$pattern	Lines ending with the pattern
\<pattern	Words starting with the pattern
pattern\>	Words ending with the pattern
-v	Lines not containing the pattern

DAY 5 - END