Session 3

Agenda

- Finding Files
- File Compression and Archiving
- Creating Links Between Files
- Redirecting Output

Finding Files

There are several commands on Linux systems that allow you to search for files. One of them :

- `find`

`find` Command Syntax

The general syntax for the `find` command is as follows:

```
find [options] [path...] [expression]
```

- The `options` attribute controls the treatment of the symbolic links, debugging options, and optimization method.
- The `path` ... attribute defines the starting directory or directories where find will search the files.
- The `expression` attribute is made up of options, search patterns, and actions separated by operators.

`find` Command Options

Command	Description
`-name pattern`	Searches for files with a specific name or pattern.
`-iname pattern`	Case-insensitive version of `-name`. Searches for files with a specific name or pattern, regardless of case.
`-type type`	Specifies the type of file to search for (e.g., `f` for regular files, `d` for directories).
`-size [+/-]n`	Searches for files based on size. `+n` finds larger files, `-n` finds smaller files. `n` measures size in characters.

Size Option:

You can use the following suffixes to specify the file size:

- b : 512-byte blocks (default)
- c : bytes
- k : Kilobytes
- M : Megabytes
- G : Gigabytes

Examples:

- To search for text files in the /home directory:

```
find /home -name "*.txt"
```

- To find empty files:

```
find /path/to/search  -empty
```

- To search for files modified within the last 7 days:

```
find /path/to/search  -mtime -7
```

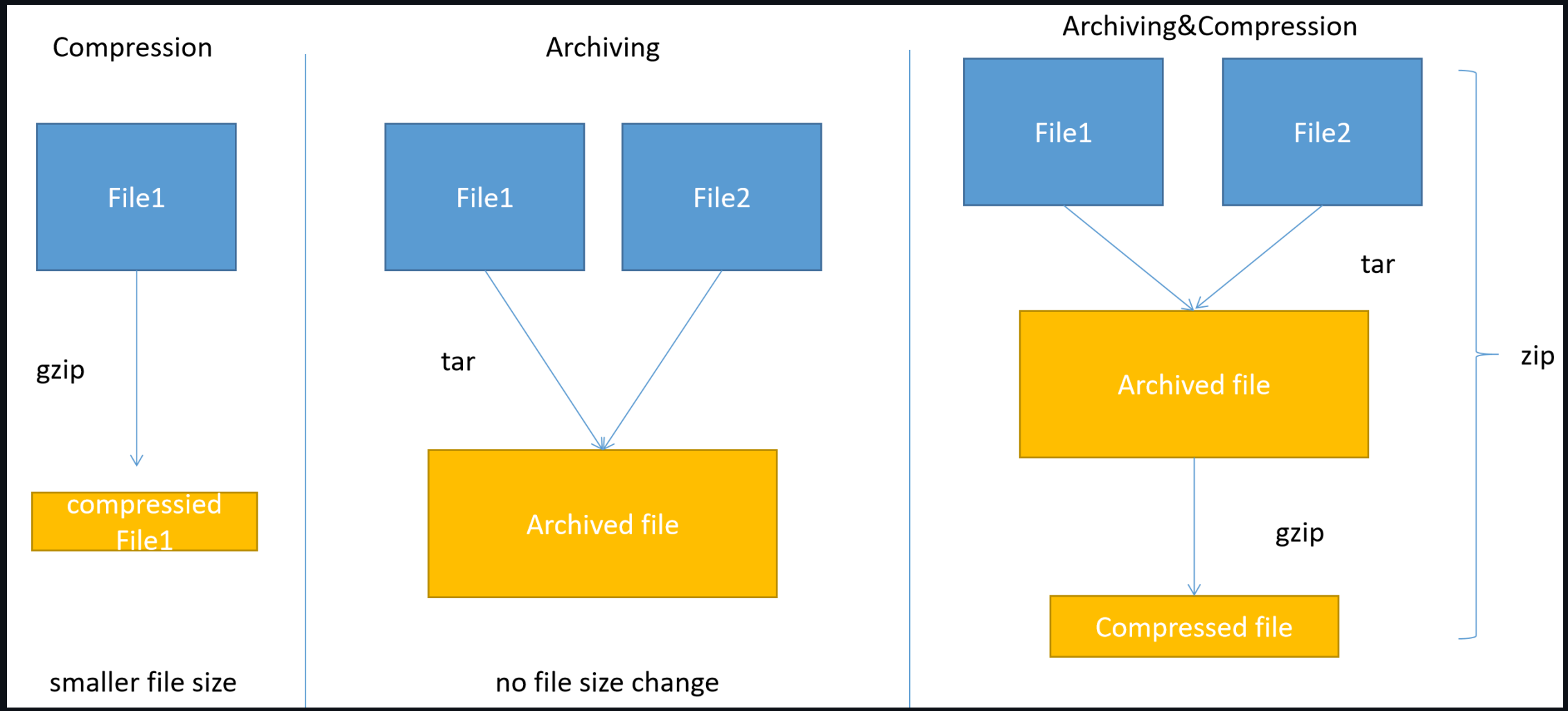
- To search for directories:

```
find /path/to/search -type d
```

File Compression and Archiving

File Compression VS File Archiving

- An **archive file** is a collection of files and directories that are stored in one file. The archive file is not compressed — it uses the same amount of disk space as all the individual files and directories combined.
- A **compressed file** is a collection of files and directories that are stored in one file and stored in a way that uses less disk space than all the individual files and directories combined. If you do not have enough disk space on your computer, you can compress files that you do not use very often or files that you want to save but do not use anymore. You can even create an archive file and then compress it to save disk space.


Compression
File1
gzip
compressied File1
smaller file size
Archiving
File1
File2
tar
Archived file
no file size change
Archiving&Compression
File1
File2
tar
Archived file
gzip
Compressed file
zip

Compressing and Decompressing Files Commands

Compressing Files Commands

`gzip`, **`bzip2`** compresses the file, adds a ".gz",".bz2" suffix, and by default deletes the original file.

- *gzip:* `gzip [Options] [filenames]`
- *bzip2:* `bzip2 [Options] [filenames]`
- *zip:* `zip [archive-name.zip] [file1 file2]`

Decompressing Files Commands

- *gunzip:* `gunzip fileName.gz`
- *bunzip2:* `bunzip2 fileName.bz2`
- *unzip:* `[archive-name.zip]`

Some of the options used with the `gzip` / `bzip2` are:

Option	Description
-k (keep)	Compress a file and keep the original file
-d (decompress)	Decompress. This causes gzip to act like `gunzip`.
-r (recursive)	Recursively compress files contained within a directory.
-1 : -9	Set ammount of compression. range from 1 (fastest, least compression) to 9 (slowest, most compression). Default value is 6.

Creating Archives with `tar` Command

What is `tar`?

```
tar [options] [archive-file] [file or directory to be archived]
```

Here ,

- tar: The command itself.
- [options]: Optional flags or settings that modify the behavior of the tar command.
- [archive-file]: The name of the archive file you are creating or working with.
- [file or directory to be archived]: The file or directory you want to include in the archive.

`tar` Options

Option	Description
-c	Creates an archive by bundling files and directories together.
-x	Extracts files and directories from an existing archive.
-f	Specifies the filename of the archive to be created or extracted.
-v	Displays verbose information, providing detailed output during the archiving or extraction process.
-z	Compresses the tar file with gzip.
-j	Compresses the tar file with bzip2.

Examples of `tar` Command Usage

1. **Creating a Simple Archive**:

This command creates a `tar` archive named `archive.tar` from the contents of `/path/to/directory`.

```
tar -cvf archive.tar /path/to/directory
```

2. **Creating a Compressed Archive with gzip**:

This creates a `gzip` compressed archive named `archive.tar.gz`.

```
tar -czvf archive.tar.gz /path/to/directory
```

3. **Extracting an Archive**:

This extracts the contents of `archive.tar` in the current directory.

```
tar -xvf archive.tar
```

4. **Extracting a Compressed Archive**:

This extracts the contents of the `gzip` compressed `archive.tar.gz`.

```
tar -xzvf archive.tar.gz
```

5. **Listing the Contents of an Archive**:

This lists all files and directories within `archive.tar` without extracting them.

```
tar -tvf archive.tar
```

Hands-On

1. Create a new directory named `compression_test` on your desktop and navigate into it. Then create three text files (`file1.txt` , `file2.txt` , and `file3.txt`).
2. Use the `nano` text editor to add different content to each file.
3. Compress all three files into a single archive using `tar` with gzip compression.
4. Decompress the archive using `tar` .
5. List the files in the directory to verify that they were decompressed correctly.
6. Compress all three files into a single archive using `tar` with bzip2 compression.

7. Decompress the bzip2 archive using `tar`.
8. List the files in the directory again to verify.
9. Locate `file1.txt` starting from the home directory (`~`).
10. Locate all files ending with `.txt` starting from the home directory (`~`).
11. Locate all files modified in the last 7 days.
12. Locate and delete all temporary files (`*.tmp`) in the `compression_test` directory.
13. Compress the entire `compression_test` directory into an archive.
14. Decompress the directory archive into a new location.

Break!😄

Creating Links Between Files

- **Links** are connections between a file name and the actual data on the disk.
- There are two main types of links:
  - **Soft Links**
  - **Hard Links**

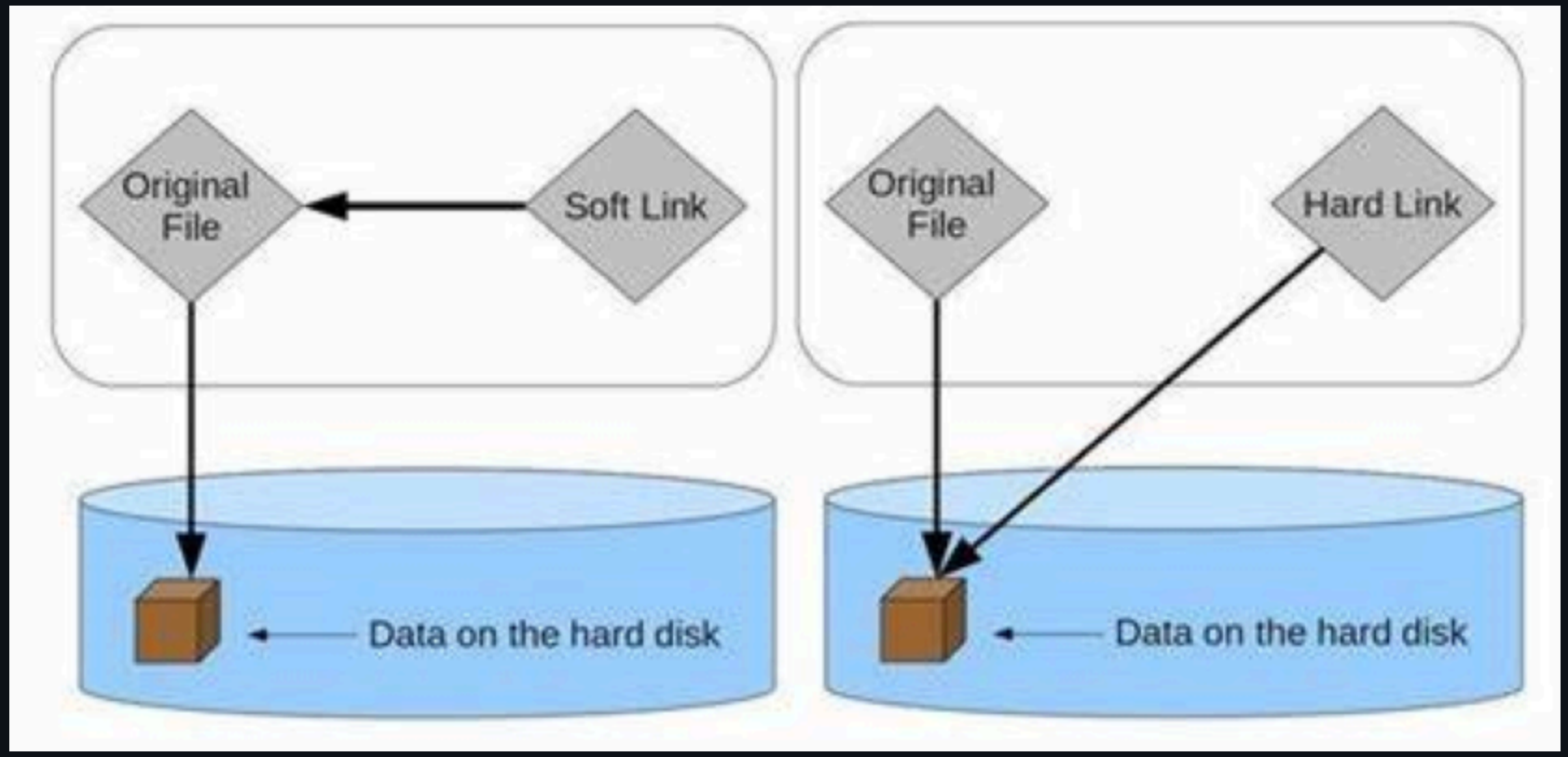
Soft Link (Symbolic Link)

- **Representation**: Points to the filename of the original file.
- **Properties**:
  - Just like "Shortcuts" in windows
  - Can cross file system boundaries.
  - Can link to directories.
  - If the original file is deleted, the soft link becomes a dangling link (broken).
- `ln -s [file path] [link path]`: The option `-s` makes the link a soft link and not a hard one.

Hard Link

- **Representation**: Points directly to the inode of the original file.
- **Properties**:
  - Cannot cross file system boundaries.
  - Cannot link to directories (on most file systems).
  - If the original file is deleted, the hard link still provides access to the file content.
- `ln [file path] [link path]`

Difference Between Soft Link and Hard Link



Importance of Links in Linux

- **File Management:**
  - Links provide efficient ways to manage and organize files.
  - They allow the same file to appear in multiple directories **without duplicating the file itself.**
- **Space Efficiency:**
  - Hard links help in saving disk space by allowing multiple references to the same file content.
- **Backup and Recovery:**
  - Hard links are used in incremental backup systems to link to unchanged files, reducing backup size and time.

Delete Links

Deleting a Soft Link (Symbolic Link):

- **Effect:** Only the symbolic link itself is deleted.
- **Original File:** The original file remains unaffected and intact.
- **Other Links:** Any other links (hard or soft) to the original file remain unaffected.

Deleting a Hard Link:

- **Effect:** The specific hard link is removed.
- **Original File:** The original file is not deleted as long as there is at least one remaining hard link to it.
- **Other Links:** Any other hard links to the file continue to function normally. The file's content remains accessible through these links.

Deleting the Original File (to which soft and hard links point):

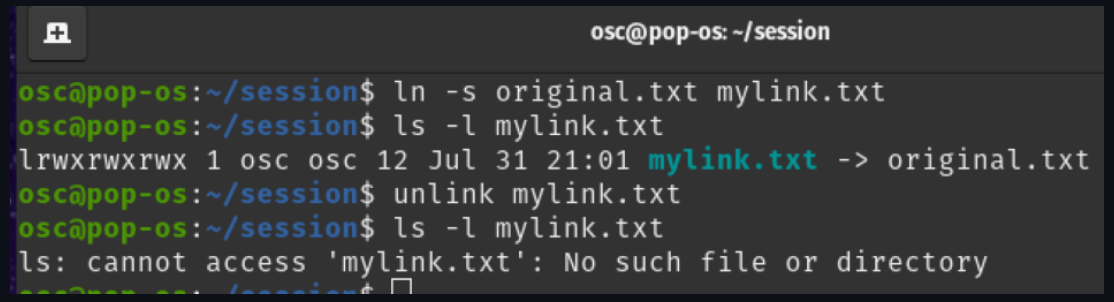
- **Soft Link:**
  - **Effect:** The soft link becomes a "dangling" link, pointing to a non-existent file. If you try to access it, you'll get an error indicating that the file does not exist.
  - **Original File:** The original file is deleted.
- **Hard Link:**
  - **Effect:** Deleting the original file doesn’t affect hard links. The file’s content remains accessible through any remaining hard links.
  - **Original File:** The concept of an "original" file is somewhat misleading with hard links because all hard links to a file are equally valid references to its data on disk. The file is only actually deleted when the last hard link is removed.

The `unlink` and `rm` commands are used to delete files, including symbolic links.

`unlink` Command

- Use `unlink` for simple, single-file deletion.

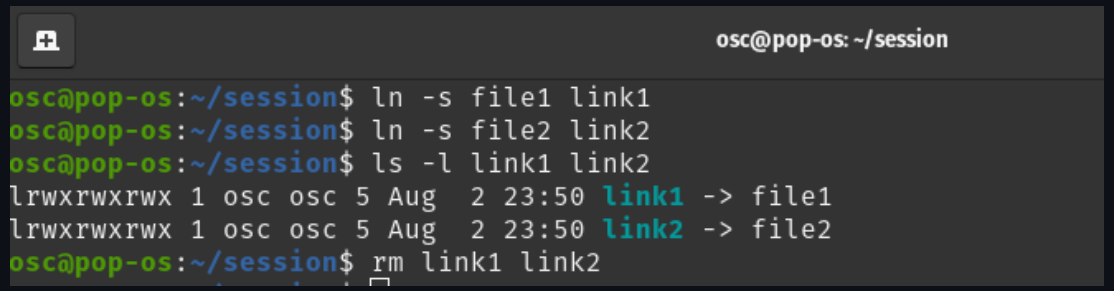
```
unlink <filename>
```



`rm` Command

- Use `rm` for **multiple files**, directories, and forced deletion.

```
rm [options] <filename>
```



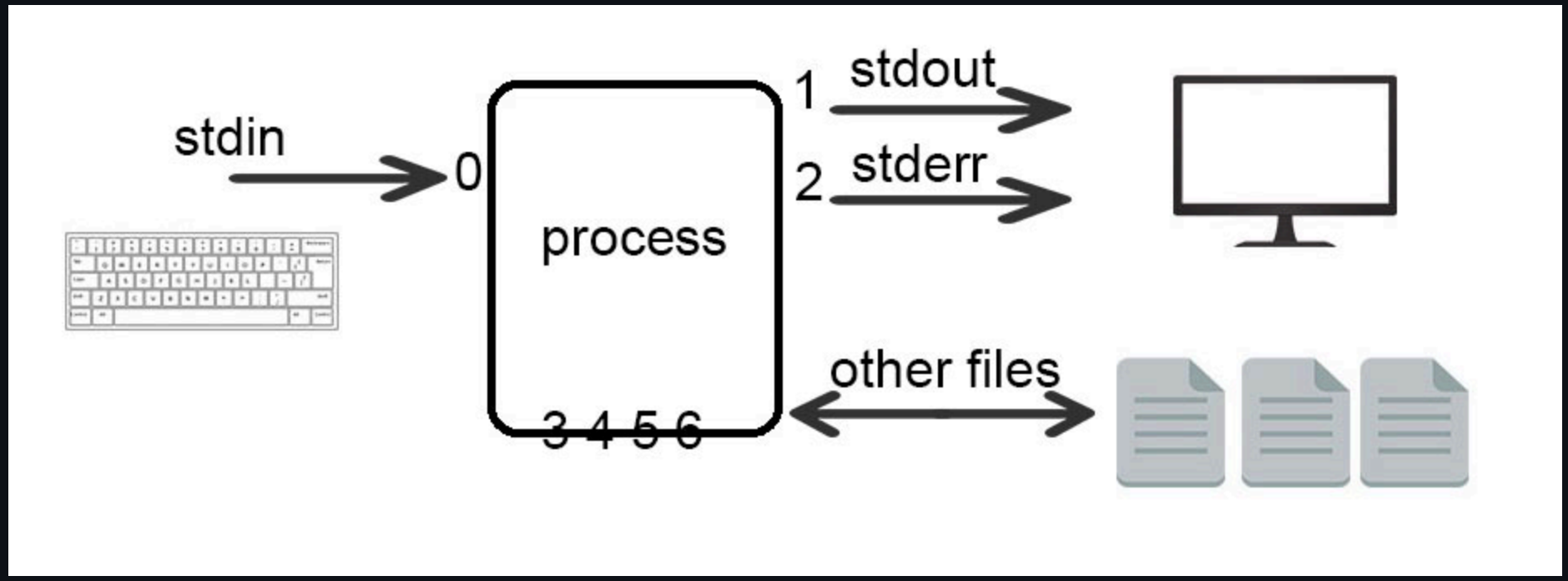
Hands on

1. Create a file named "**original.txt**" and write some content into it.
2. Create a hard link for "**original.txt**" named "**copy.txt**"
3. change content of "**copy.txt**" then check if the content of "**original.txt**" is changed
4. Delete "**original.txt**" and use `ls -l` ,see what happened to the link
5. echo the content of "**copy.txt**"

Redirecting Output to a File

Standard Input , Output , Error

A process structure is constructed with numbered channels (file descriptors) to manage open files. Processes connect to files to reach data content or devices these files represent. Processes are created with default connections for channels **0, 1, and 2**, known as standard **input**, standard **output**, and standard **error**. Processes use channels **3 and above** to connect to **other files**.



Channels

NUMBER	CHANNEL NAME	DESCRIPTION	DEFAULT CONNECTION	USAGE
0	stdin	Standard input	Keyboard	read only
1	stdout	Standard output	Terminal	write only
2	stderr	Standard error	Terminal	write only
3+	filename	Other files	none	read and/or write

Input/Output Redirection

we can redirect **Standard Output (stdout)** into file by using the greater than sign ">"

```
$ echo "Hello World"
Hello World
$ echo "Hello World" > myfile.txt
$ cat myfile.txt
Hello World
$ echo "Hello World 1" > myfile.txt
$ cat myfile.txt
Hello World 1
$ echo "Hello World 2" >> myfile.txt
$ cat myfile.txt
Hello World 1
Hello World 2
```

I/O Redirection

we can redirect **Standard Error (stderr)** into file by using the sign **"2>"**

```
$ cat blabla
cat: blabla: No such file or directory
$ cat blabla > errorfile.txt
cat: blabla: No such file or directory
$ cat errorfile.txt
$ cat blabla 2> errorfile.txt
$ cat errorfile.txt
cat: blabla: No such file or directory
$
```

Notes

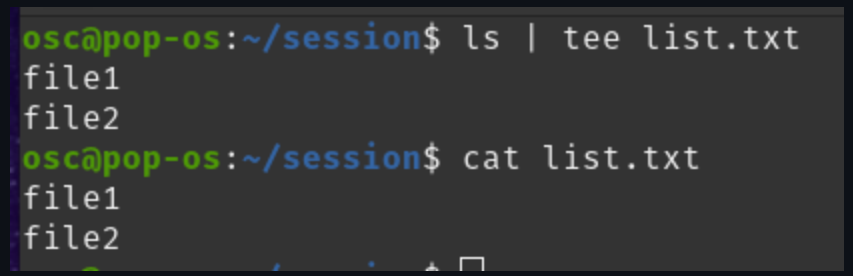
- ">" for override
- ">>" for append without deleting any thing

Pipeline

what the pipe does is that takes any output passed to it then uses it as an input for the following command.

Pipelines, and the `tee` Command

Display the output of a command on the terminal (standard output) and Write the same output to a file you specify.

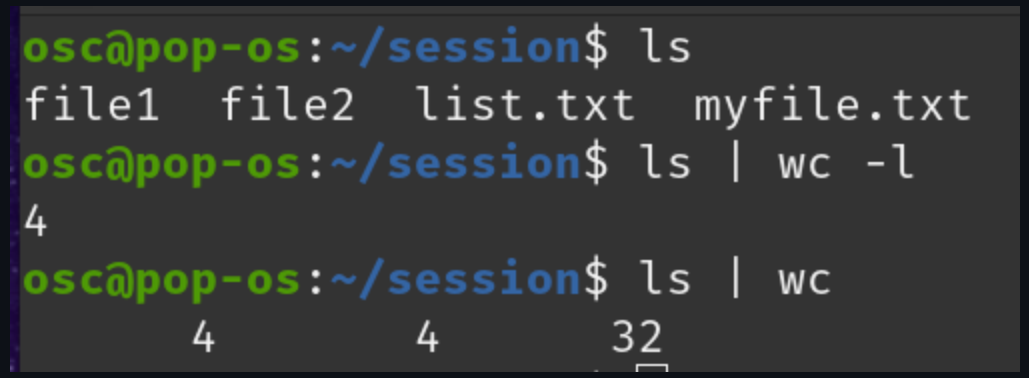


Explanation

- `ls` : Lists the contents of the current directory.
- `|` : Pipes the output of ls to the next command.
- `tee list.txt` : Writes the output to list.txt and also prints it on the terminal.

Pipelines, and the `wc` Command

The `wc` command counts lines, words, and characters in a file. It takes a ***-l***, ***-w***, or ***-c*** option to display only the number of lines, words, or characters, respectively.



Explanation

- `ls` : Lists the contents of the current directory.
- `|` : Pipes the output of ls to the next command.
- `wc -l` : Counts the number of lines in the output, which corresponds to the number of items in the directory.

Examples for `wc` and `tee` Commands

1. **Writing the output to a file and displaying it on the terminal**:

   ```
   echo "Hello, World!" | tee output.txt
   ```

   This command will write "Hello, World!" to `output.txt` and also display it on the terminal.

2. **Counting lines, words, and characters in multiple files**:

   ```
   wc file1.txt file2.txt
   ```

   This command will output the number of lines, words, and characters for each file and the total for all files.

3. **Piping the output of a command through `tee` to another command:**

```
echo "Hello, World!" | tee >(wc -w)
```

This command will display "Hello, World!" on the terminal, count the words (output by `wc -w`), and display the word count.

Hands on

1. Create a file named **output.txt**
2. redirect output of this command `echo "Hello!"` to **output.txt** file
3. redirect output of this command `ls -l` to be appended to **output.txt** file
4. count the number of files and directories in your current directory

Thanks

Any Questions?