

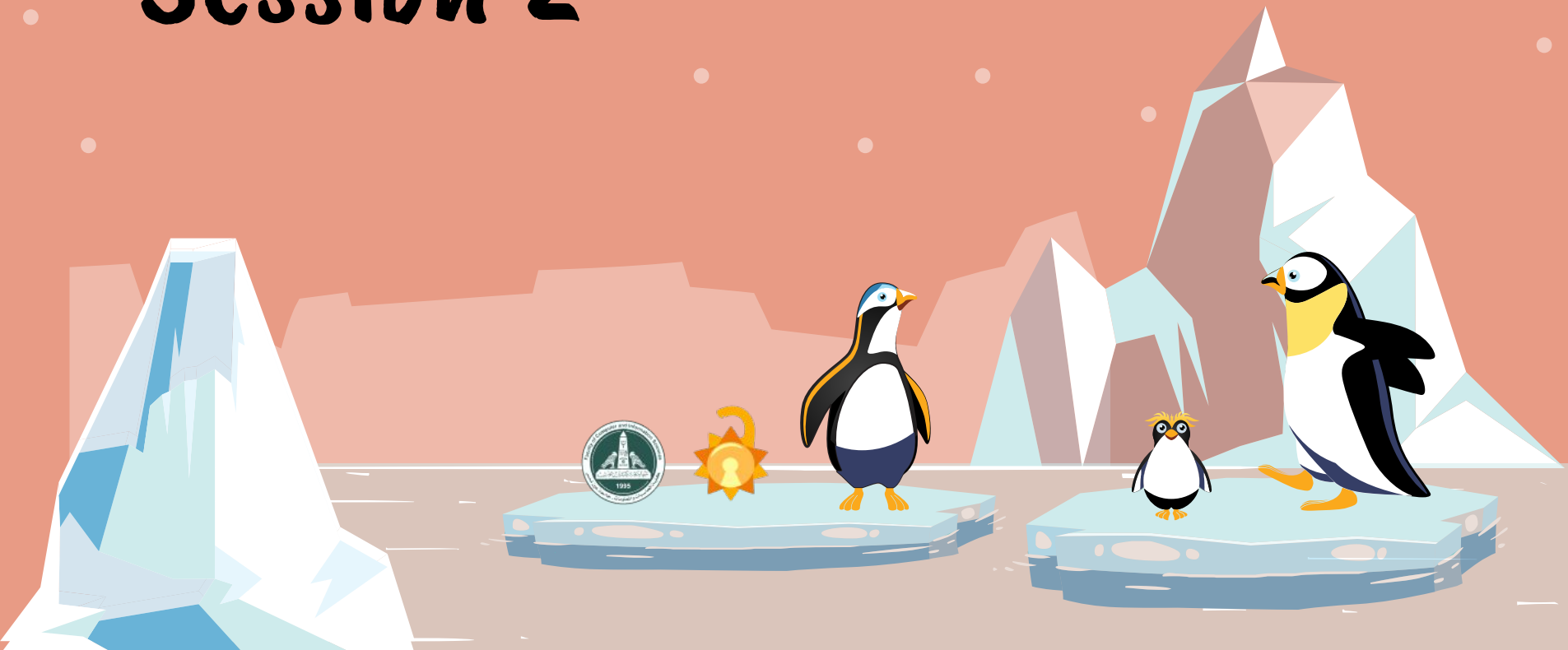


# Linux Summer Training 23





# Session 2





# Agenda

**01**

*Compression and Archiving*

**03**

*Links Between Files*

**02**

*Finding Files*

**04**

*Redirecting Output*



# Compression and Archiving

- An **archived file** is a collection of files and directories that are stored in one file
- 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



# Archiving using tar



# tar modes

- **-c** create a new archive
- **-x** extract files from an archive
- **-t** list the contents of an archive
- **-v** verbose
- **-z** compress the tar file with gzip
- **-j** compress the tar file with bzip2



# Compress



# Decompress

gzip



gunzip



bzip2



bunzip2



zip



unzip





# gzip/bzip2 flags

- **-c** Write output to standard output and keep the original file
- **-f** force compression
- **-l** list compression statistics (gzip only)
- **-r** Recursively compress (gzip only)
- **-v** verbose





# More flags

- . -1 Fast
- . -9 Best



# Agenda

**01**

**Compression and Archiving**

**03**

**Links Between Files**

**02**

**Finding Files**

**04**

**Redirecting Output**



# Finding Files





# Find command

-name

-user

-empty

-perm

-size



# Size options

b ☐ 512 byte blocks

c ☐ bytes

k ☐ Kilobytes

m ☐ Megabytes

g ☐ Gigabytes



```
$ find . -name work-bench  
./work-bench
```

```
$ find . -iname Work-bench  
./work-bench
```

```
$ find ~/Desktop -size 4k  
/home/suhail/Desktop  
/home/suhail/Desktop/workspace
```

# Hands on

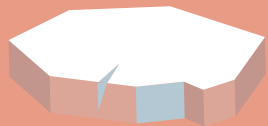


1. make 2 files (file1.txt / file2.txt) on your desktop
2. edit them using nano and both files should have different content
3. compress file1 with whatever tool you want to
4. decompress the same file with the same tool
5. use find to locate that file.txt from parent directory "~"
6. use find to locate all files that end with .txt





# Agenda



**02**

*Finding Files*

**03**

*Links Between Files*

**04**

*Redirecting Output*

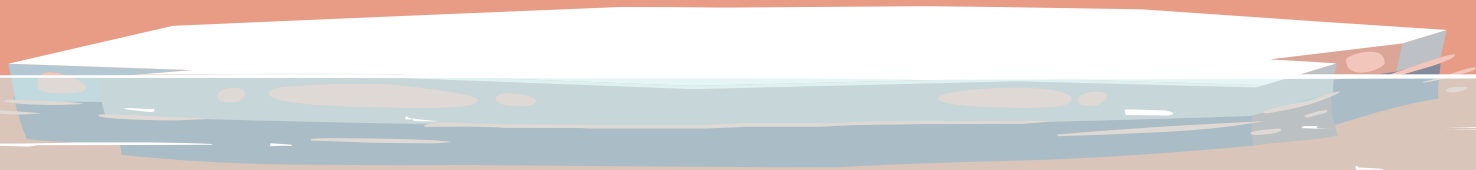


**BREAK**





# Links





Inode



# Information contained in an inode

- File size
- Device on which the file is stored
- User and group IDs associated with the file
- Permissions needed to access the file
- Creation, read, and write timestamps



ls -li



# Symbolic/Soft Links

- Just like 'Shortcuts' in Windows
- Just a pointer to the original file/directory
- Different inode from the source file
- Renaming, deleting or moving the source could corrupt the link

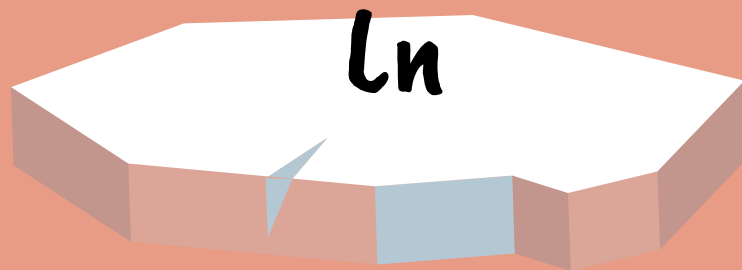


$\ln -s$





# Hard Links





# Hard Links

- Just like a copy but with a connection between the files
- Works only with files not directories
- Same inode as the source file
- Renaming, deleting or moving the source does not affect the link
- Moving the link across file systems could corrupt the link

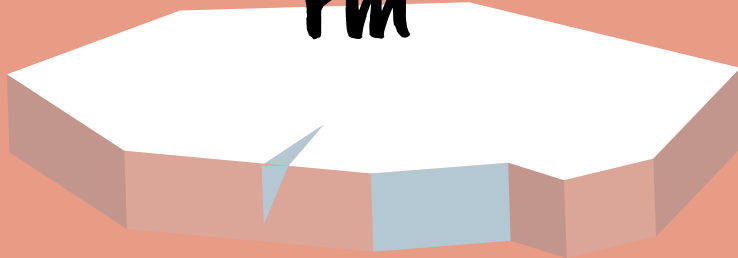


# Deleting Links

**unlink**



**rm**

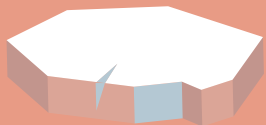




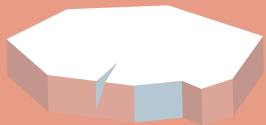
# Dangling Soft Link



# Agenda



*Links Between Files*



*Redirecting Output*




# Redirecting Output



- . Standard input (channel 0) reads input from the keyboard.
- . Standard output (channel 1) sends normal output to the terminal.
- . Standard error (channel 2) sends error messages to the terminal.





Running "ls" gives normal  
output, thus channeled to  
stdout

```
satharus@Argon: ~/Arduino$ ls  
libraries  
satharus@Argon: ~/Arduino$ |
```



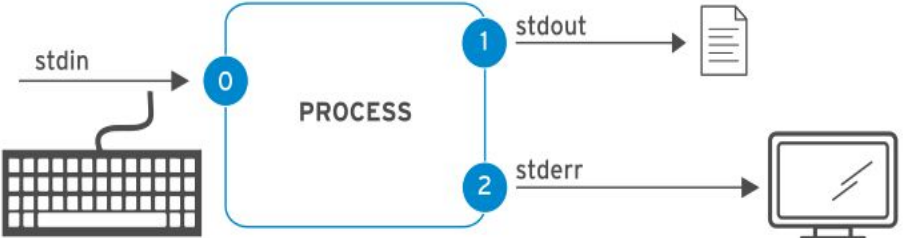
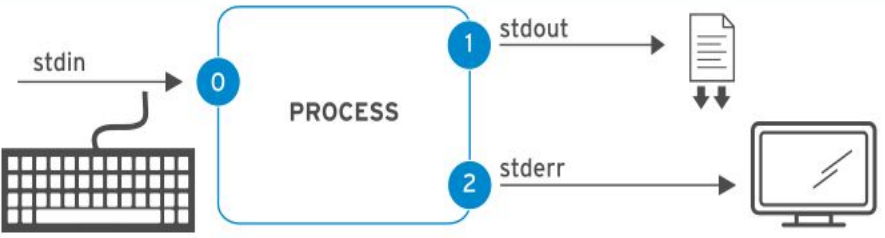
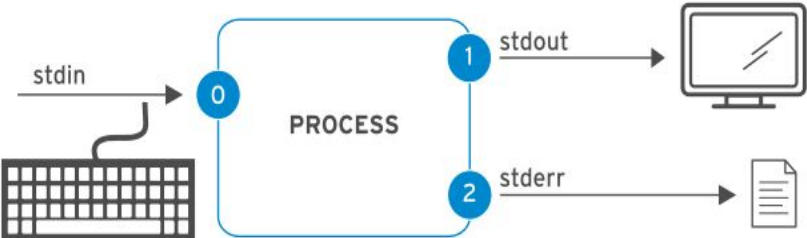
Running hello or any unknown command or syntax will give an error, thus channeled to stderr

```
satharus@Argon: ~/Arduino$ hello  
bash: hello: command not found  
satharus@Argon: ~/Arduino$ |
```

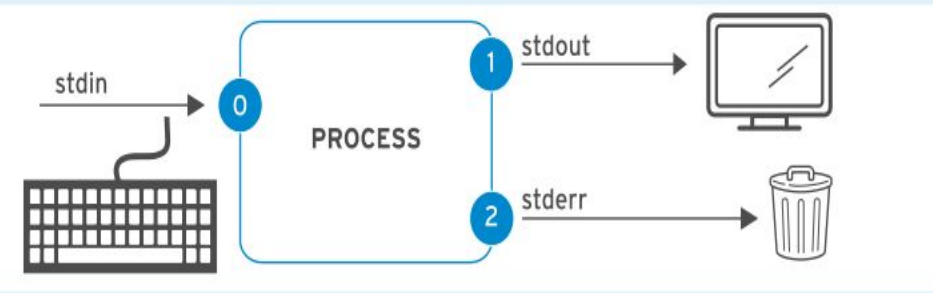
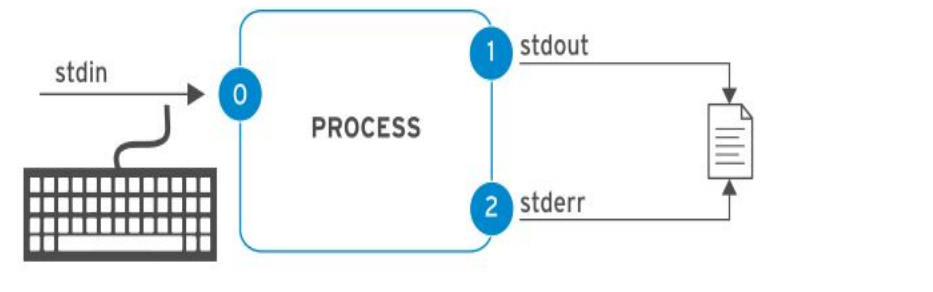
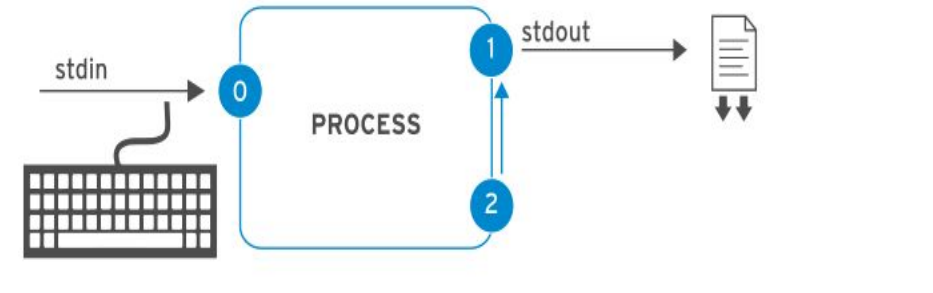



# Redirecting Output to a File




USAGE	EXPLANATION	VISUAL AID
<code>&gt; file</code>	redirect <b>stdout</b> to overwrite a file	
<code>&gt;&gt; file</code>	redirect <b>stdout</b> to append to a file	
<code>2&gt; file</code>	redirect <b>stderr</b> to overwrite a file	



<code>2&gt; /dev/null</code>	discard <b>stderr</b> error messages by redirecting to <b>/dev/null</b>	
<code>&gt; file 2&gt;&amp;1</code>	redirect <b>stdout</b> and <b>stderr</b> to overwrite the same file	
<code>&gt;&gt; file 2&gt;&amp;1</code>	redirect <b>stdout</b> and <b>stderr</b> to append to the same file	



Why error  
redirection?





# I/O Redirection

We can redirect  
stdout into file  
by using the  
greater than  
sign ">"

```
satharus@Argon:~/Arduino$ ls
libraries
satharus@Argon:~/Arduino$ ls > output.txt
satharus@Argon:~/Arduino$ cat output.txt
libraries
output.txt
satharus@Argon:~/Arduino$ ls
libraries output.txt
satharus@Argon:~/Arduino$ |
```



# I/O Redirection

We can  
redirect stderr  
into file by  
using the  
sign 2>.

```
satharus@Argon:~/Arduino$ hello
bash: hello: command not found
satharus@Argon:~/Arduino$ hello > output2.txt
bash: hello: command not found
satharus@Argon:~/Arduino$ cat output2.txt
satharus@Argon:~/Arduino$ hello 2> err.txt
satharus@Argon:~/Arduino$ cat err.txt
bash: hello: command not found
satharus@Argon:~/Arduino$ |
```





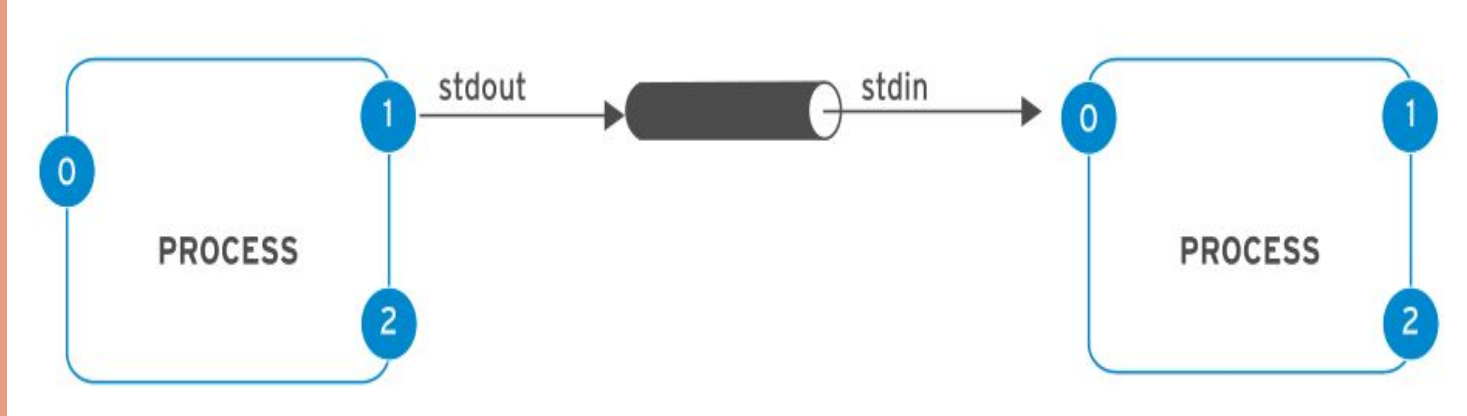
# Notes

- Using `>` will override what's in the file
- To append to the file without deleting its content we can add another greater than sign `>>`
- We can also use `&>` to redirect both `STDOUT` and `STDERR`.



# Pipeline

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





You can do all sort of  
fun things with pipes.  
For example:  
`fortune | cowsay |  
lolcat`

```
osc@mint:~$ fortune | cowsay | lolcat

/ Hope you are having fun learning about \
\ Linux with OSC                          /
-----
      ^ ^
      (oo)\_____/
      (_____)  /\
              ||----w |
              ||     ||
```



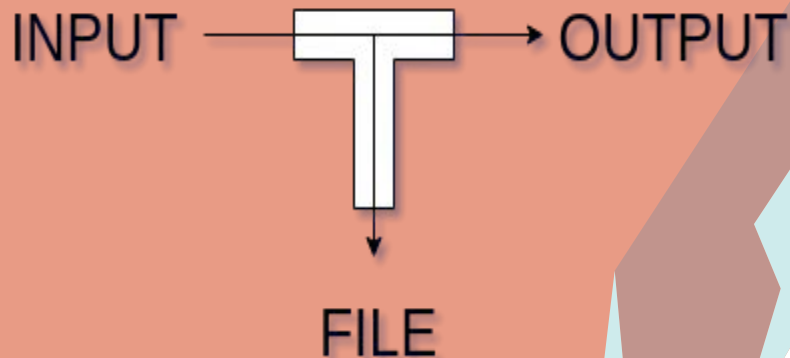
# Pipeline examples

- `ls -l /usr/bin | less`
- `ls | wc -l`
- `ls | head -n 10 > /tmp/ten-last-changed-files`



# Pipelines, Redirection, and the tee Command

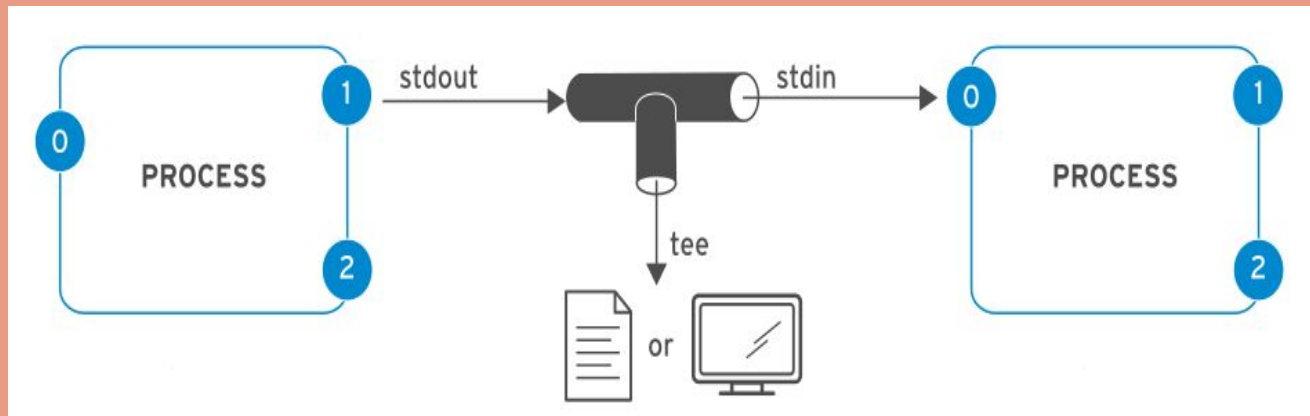
If you imagine data as water flowing through a pipeline, tee can be visualized as a "T" joint in the pipe which directs output in two directions.





# Pipeline Examples Using the tee Command

```
ls -l | tee /tmp/saved-output | less
```





# 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.
- You can use it with pipe to count number of files in a directory

# Hands on





1. make a hard link to file2
2. prove that the files are linked (Hint : use ls and its flags)
3. echo "Redirecting output" in a file1.txt that you have made
4. append to the same file your name using echo
5. delete file1 and use ls -l ,observe what happened to the link
6. delete file2 and see the content of the hard link
7. count the number of the files existing in home using pipeline
8. use find to locate all empty files and pipeline the output to a command for interactive reading



# Thanks!

