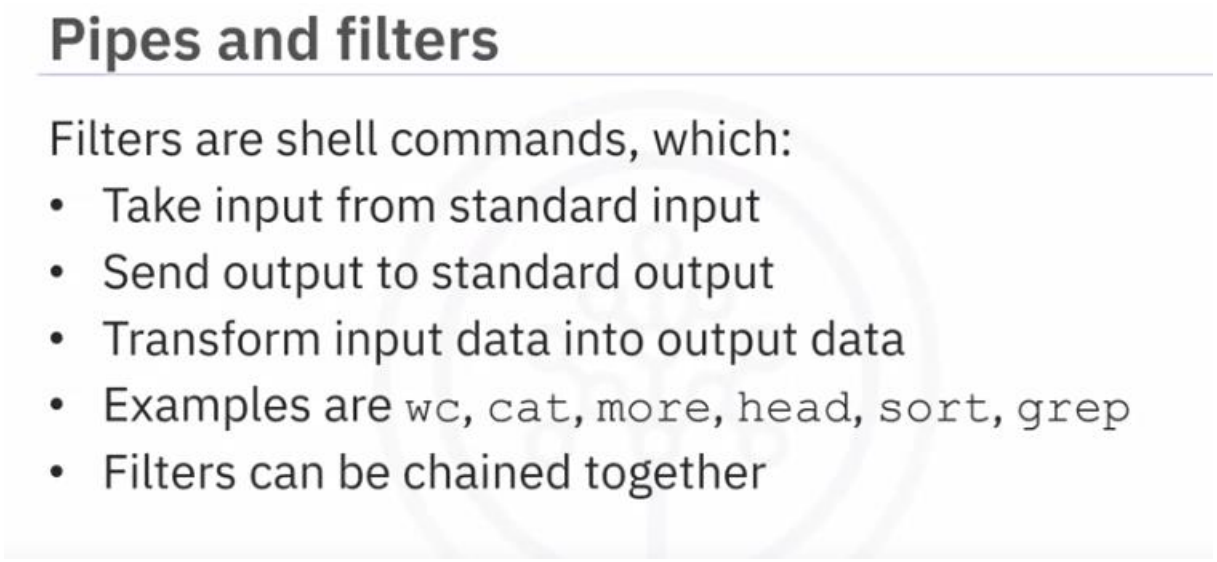
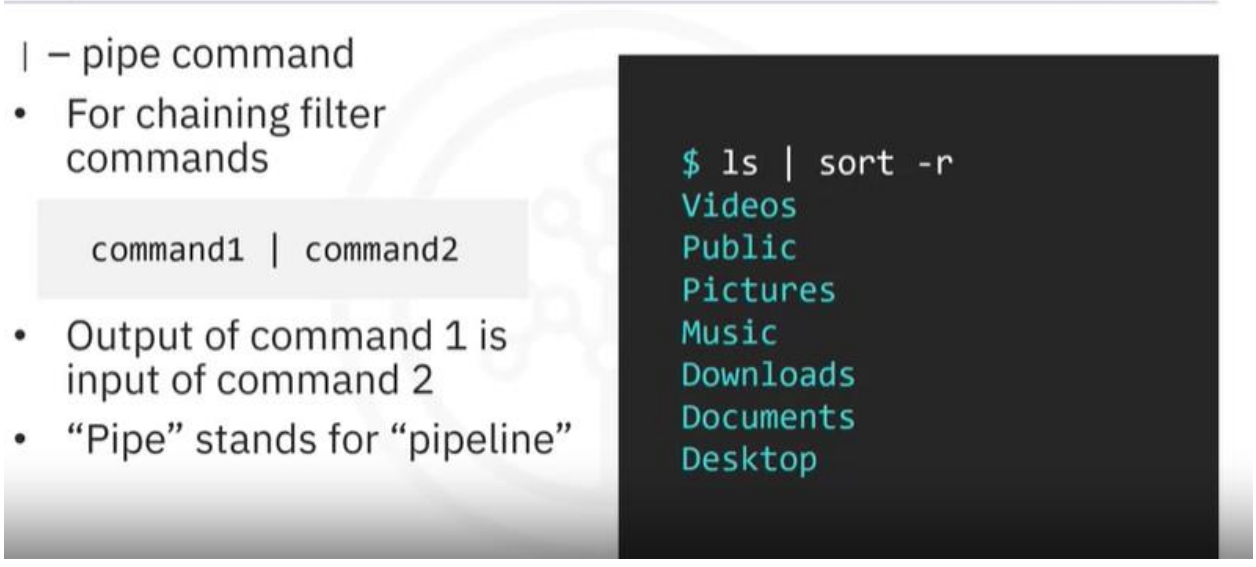
Shell Scripting Basics

- A shell script is a program that begins with a “shebang” directive
- Shell scripts are used to run commands and programs
- Scripting languages are interpreted, not compiled
- Compiled languages faster but require longer time

Filters, Pipes and Variables



- | -> pipe command



- Shell variables:

  + Define shell variables: varname = value

```
$ GREETINGS="Hello"
$ echo $GREETINGS
Hello

$ AUDIENCE="World"
$ echo $GREETINGS $AUDIENCE
Hello World

$ unset AUDIENCE
```

- Environment variables -> Extend scope
  export var_name

Useful features of Bash shell

- # -> comment
- ; -> command separator
- * -> filename expansion wildcard
- ? -> single character wildcard in filename expansion
- \ -> escape unique character interpretation
- > -> Redirect output to the file
- >> -> Append output to the file
- 2> -> Redirect standard error file
- 2 >> -> Append std error file
- < -> Redirect file contents to standard input

Command substitution

- Replace the command with its output
  `$(command)` or `` `command` ``
- Store output of `pwd` command in `here`:

```
$ here=$(pwd)
$ echo $here
/home/jgrom
```

- Batch mode: Run commands sequentially
  Command1, command2

- Concurrent mode: Commands run in parallel
  Command 1 & command 2

Advanced bash scripting

- Conditionals: if – then – else syntax

```
if [ condition ]
then
    statement_block_1
else
    statement_block_2
fi
```

- *You must always put spaces around your condition within the square brackets `[ ]`.*
- *Every `if` condition block must be paired with a `fi` to tell Bash where the condition block ends.*
- *The `else` block is optional but recommended. If the condition evaluates to `false` without an `else` block, then nothing happens within the `if` condition block. Consider options such as echoing a comment in `statement_block_2` to indicate that the condition was evaluated as `false`.*

- Logical operators: “==”, “!=”

Arithmetic calculations: Every calculations must be put inside $(())

```
echo $((3+2))
```

or

```
a=3
b=2
```

```
my_array=(1 2 "three" "four" 5)
```

This statement creates and populates the array `my_array` with the items in the parentheses: `1`, `2`, `"three"`, `"four"`, and `5`.

You can also create an empty array by using:

```
declare -a empty_array
```

- Array:

If you want to add items to your array after creating it, you can add to your array by appending one element at a time:

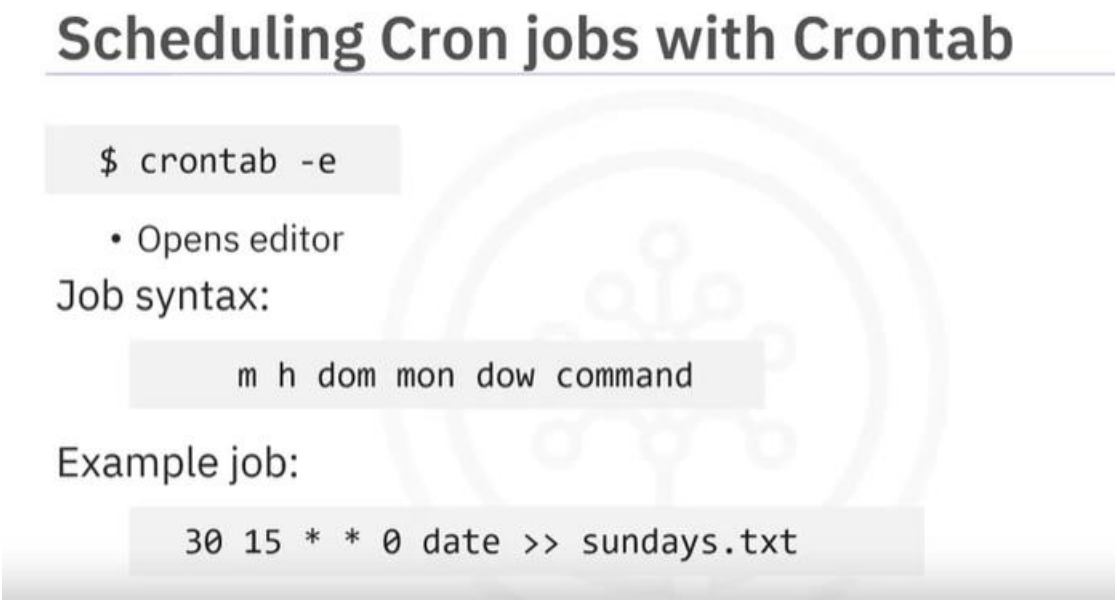
```
my_array+=("six")
my_array+=(7)
```

Indexing similar to array in python

```
for i in ${!my_array[@]}; do
  echo ${my_array[$i]}
done
```

The `for` loop requires a `; do` component in order to cycle through the loop. Additionally, you need to terminate the `for` loop block with a `done` statement.

Scheduling Jobs using Cron



Viewing and removing Cron jobs

```
jgrom@GROOT617:~$ crontab -l  | tail -6
#
# m  h   dom mon dow  command

  30 15  *   *   0    date >> path/sundays.txt
   0  0  *   *   *    /cron_scripts/load_data.sh
   0  2  *   *   0    /cron_scripts/backup_data.sh
jgrom@GROOT617:~$
```