**Managing files and directories**

* **cd** directory: changes the current working directory to the specified one
* **pwd:** prints the current working directory
* **ls:** lists the contents of the current directory
* **ls** directory: lists the contents of the received directory
* **ls** -l: lists the additional information for the contents of the directory
* **ls** -a: lists all files, including those hidden
* **ls** -la: applies both the -l and the -a flags
* **mkdir** directory: creates the directory with the received name
* **rmdir** directory: deletes the directory with the received name (if empty)
* **cp** old\_name new\_name: copies old\_name into new\_name
* **mv** old\_name new\_name: moves old\_name into new\_name
* **touch** file\_name: creates an empty file or updates the modified time if it exists
* **chmod** modifiers files: changes the permissions for the files according to the provided modifiers; we've seen +x to make the file executable
* **chown** user files: changes the owner of the files to the given user
* **chgrp** group files: changes the group of the files to the given group

**Operating with the content of files**

* **cat** file: shows the content of the file through standard output
* **wc** file: counts the amount of characters, words, and lines in the given file; can also count the same values of whatever it receives via stdin
* **file** file: prints the type of the given file, as recognized by the operating system
* **head** file: shows the first 10 lines of the given file
* **tail** file: shows the last 10 lines of the given file
* **less** file: scrolls through the contents of the given file (press "q" to quit)
* **sort** file: sorts the lines of the file alphabetically
* **cut -d**separator **-f**fields file: for each line in the given file, splits the line according to the given separator and prints the given fields (starting from 1)

**Additional commands**

* **echo** "message": prints the message to standard output
* **date:** prints the current date
* **who:** prints the list of users currently logged into the computer
* **man** command: shows the manual page of the given command; manual pages contain a lot of information explaining how to use each command (press "q" to quit)
* **uptime:** shows how long the computer has been running
* **free:** shows the amount of unused memory on the current system

### Managing streams

These are the redirectors that we can use to take control of the streams of our programs

* command **>** file: redirects standard output, overwrites file
* command **>>** file: redirects standard output, appends to file
* command **<** file: redirects standard input from file
* command **2>** file: redirects standard error to file
* command1 **|** command2: connects the output of command1 to the input of command2

### Operating with processes

These are some commands that are useful to know in Linux when interacting with processes. Not all of them are explained in videos, so feel free to investigate them on your own.

* **ps:** lists the processes executing in the current terminal for the current user
* **ps** ax: lists all processes currently executing for all users
* **ps** e: shows the environment for the processes listed
* **kill** PID: sends the SIGTERM signal to the process identified by PID
* **fg**: causes a job that was stopped or in the background to return to the foreground
* **bg:** causes a job that was stopped to go to the background
* **jobs:** lists the jobs currently running or stopped
* **top:** shows the processes currently using the most CPU time (press "q" to quit)

CURSO: https://ryanstutorials.net/bash-scripting-tutorial/bash-variables.php

>> Para hacer ejecutable un archivo:

**chmod +x <file.extension>**

Luego Podemos ejecutarlo con:

**./file.extension**

>> Para mirar la información de un archivo (y ver si es ejecutable)

**file <FILENAME>**

**VARIABLES**

Van precedidas de un $ para leerlas.

El $ no es necesario para crearlas.

Podemos referirnos a los INPUTS de un Script usando $ seguido del número…

Texto

Descripción generada automáticamente

Sí funciona (primero debimos crear el segundo archivo):

Texto

Descripción generada automáticamente

Otras variables son:

* $0 – Nombre del script
* $1 - $9 – Los primeros 9 argumentos ingresados al script
* $# - Número de argumentos que pasamos al script
* $@ - Todos los argumentos que pasemos
* $? – El “exit status” del último proceso ejecutado (0 significa que corrió con éxito)
* $$ - El Id del Script actual
* $USER – El username del usuario que está ejecutando el script
* $HOSTNAME – El hostname de la máquina donde se ejecuta el script
* $SECONDS – El número de segundos desde que el script comenzó a ejecutarse
* $RANDOM – Retorna diferentes números aleatorios cada que se ejecuta
* $LINENO – Retorna la línea del script actual

El comando (**ENV**) sirve para ver una lista de otras variables a las que nos podemos referir…

Podemos crear nuestras propias variables (sin espacios antes y después del =):

**variable=value**

Texto

Descripción generada automáticamente

Cuando requerimos almacenar más de una palabra, debemos usar comillas (**variable=‘Hello world’**)

Se pueden usar comillas simples o dobles; estas últimas permiten substituciones, las primeras no.

Texto

Descripción generada automáticamente

**COMMAND SUBSTITUTION**

Nos permite guardar el resultado de un comando en una variable. El comando debe estar encerrado en paréntesis, precedido de $

Texto

Descripción generada automáticamente

EXPORTING VARIABLES

Nos permite que las variables que creamos en un script puedan ser utilizadas en otro.

Texto

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

Al correr el script1, NÓTESE que el valor de las variables, en el último “echo” no cambió. Esto porque aquí no importa lo que hagan los procesos hijos con las variables, esto no afecta el valor del script original.

**USER INPUT**

Utilizamos “read <varname>” para solicitar una variable a un usuario.

Captura de pantalla de un celular

Descripción generada automáticamente

Podemos alterar el comportamiento de “READ” con algunas opciones, como pueden ser ‘-p’ que permite especificar un mensaje y ‘-s’ que hace el input silencioso (es decir, que no se muestre en pantalla).

Captura de pantalla de un celular

Descripción generada automáticamente

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

Se puede ingresar MÁS DE UNA variable:

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

Texto

Descripción generada automáticamente

LEYENDO DESDE STDIN

Nos sirve para cuando construimos pipelines en Bash:

Una captura de pantalla de un celular

Descripción generada automáticamente

En el script anterior los pipes se forman con | (notar el CAT). También la entrada se define con “/dev/stdin”.

En el intermedio se le dice que particione la entrada por el delimitador (espacios) y que tome los campos (2 y 3).

Texto

Descripción generada automáticamente

Notar nuevamente el CAT para crear el pipe.

**IF STATEMENT**

Condicionales (la misma teoría de otros lenguajes)

Captura de pantalla de un celular

Descripción generada automáticamente

Captura de pantalla de un celular con texto

Descripción generada automáticamente

**PRUEBAS:**

Tabla

Descripción generada automáticamente

Texto

Descripción generada automáticamente

También podemos hacer IF anidados, IF – ELSE, IF – ELIF – ELSE…

Imagen que contiene Texto

Descripción generada automáticamente

HAY UN CASE….!!!!!!

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**LOOPS/FUNCTIONS/USER\_INTERFACE**

Esos tres eran los siguientes temas en el enlace: <https://ryanstutorials.net/bash-scripting-tutorial/bash-variables.php>