# Introducción a BASH

Lic. Marcos Mazzini - CCAD / UNC
CPA CONICET

# Módulo 2: Uso cotidiano y trucos

# Variables de entorno

En una consola además de ingresar comandos también podemos definir **variables**.

Éstas sirven para almacenar algún valor temporal durante nuestra sesión o exportarse para **compartir configuraciones** entre aplicaciones y procesos LINUX.

# Variables de entorno

Si solo **asignamos** la variable su alcance está acotado al proceso actual (consola actual), si además la **exportamos** su valor estará disponible para todos los procesos que corramos desde esta terminal.

# comandos: =, <$VARIABLE>, export, env

```
[usuario@localhost ~]$ C1=/home/usuario/curso1
[usuario@localhost ~]$ echo $C1
/home/usuario/curso1
[usuario@localhost ~]$ cd $C1
[usuario@localhost curso1]$ pwd
/home/usuario/curso1
[usuario@localhost curso1]$ export C1
[usuario@localhost curso1]$ env
SSH_AGENT_PID=1703
HOSTNAME=localhost.localdomain
TERM=xterm-256color
SHELL=/bin/bash
HISTSIZE=1000
C1=/home/usuario/curso1
```

# Variables de entorno: env

```
OLDPWD=/home/usuario
USER=usuario
LS_COLORS=rs=0:di=38;5;27:ln=38;5;51:mh=44;38;5;15:pi=40;38;5;11:so=38;5;13:do=38;5;5:bd
=48;5;232;38;5;11:cd=48;5;232;38;5;3:or=48;5;232;38;5;9:mi=05;48;5;232;38;5;15:su=48;55:
<…>
PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/home/usuario/.local/bin:/ho
me/usuario/bin
MAIL=/var/spool/mail/usuario
DESKTOP_SESSION=mate
PWD=/home/usuario/curso1
LANG=es_AR.utf8
HOME=/home/usuario
LOGNAME=usuario
_=/usr/bin/env
```

# comandos: export <var>=<valor>, printenv

```
[usuario@localhost curso1]$ export C2=/home/usuario/curso2
[usuario@localhost curso1]$ printenv C2
/home/usuario/curso2
[usuario@localhost curso1]$ cd $C2
[usuario@localhost curso2]$
```

# Variables de entorno

Hay variables de entorno que **dependen del usuario**, si ejecutamos env como usuario o como root podemos ver diferencias.

# La variable Path

- Esta variable de entorno es la que define la **ubicación** de los comandos.
- Es una lista de ubicaciones que está inicializada con las establecidas por defecto (/usr/local/bin, /usr/bin, /bin, ...).
- Cuando escribimos un comando se busca en esas ubicaciones para ejecutarlo.
- Evita tener que escribir la ruta completa para ejecutar comandos.

# comandos: “ ” , ${ } (manejo de espacios)

```
[usuario@localhost ~]$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/home/usuario/.loca
l/bin:/home/usuario/bin
[usuario@localhost ~]$ cat scripts/saludar.sh
#!/bin/bash
echo "Hola Mundo!"
[usuario@localhost ~]$ export PATH="${PATH}:~/scripts"
[usuario@localhost ~]$ saludar.sh
Hola Mundo!
```

# Alias

Bash permite al usuario poner un **nombre** a un comando con argumentos y flags.

Podemos acortar comandos muy largos o que usamos muy frecuentemente o salvar errores que cometemos reiterados.

```
$alias nombre="<comando a ejecutar>"
```

- Las comillas evitan errores por espacios en blanco
- NO agregar espacios alrededor del =

# comandos: alias, which

```
[usuario@localhost ~]$ alias mendieta="ssh mazzini@mendieta.ccad.unc.edu.ar"
[usuario@localhost ~]$ mendieta

[mmazzini@mendieta ~]$ exit
[usuario@localhost ~]$ which ll
alias ll='ls -l --color=auto'
    /usr/bin/ls
```

# Globs y Brace Expansion

Son formas de escribir **comodines** o **permutaciones** cuando los argumentos a los comandos tienen alguna estructura y queremos evitar escribirlos todos.

Se dice que Bash **expande** las expresiones **antes** de invocar a los comandos, de forma que el comando solo recibe los argumentos, no debe procesar las expresiones.

# Globs (pathname expansion)

Globs: Expande a **nombres de archivos** que coinciden con el patrón

- * : cualquier cadena, incluida la nula
- ? : cualquier letra
- [...] : cualquiera de los caracteres incluidos entre los corchetes

# comandos: ?, *, [] (globs)

```
[usuario@localhost ~]$ ls c?rso[12]
curso1:
texto1.txt  texto2.txt

curso2:
algo.txt  nada  un archivo

[usuario@localhost ~]$ cat curso1/*.txt
contenido del archivo
texto1.txt
FIN
contenido del archivo
texto2.txt
FIN
```

# Brace Expansion

Brace Expansion: Expande a **todas** las permutaciones

- {string1,string2,...,stringN} : secuencia **a e i o u**
- {<inicio>..<fin>} : secuencia **1 2 3 4 5**
- {<inicio>..<fin>..<incr>} : con incremento **2 4 6 8 10**
- <prefijo>{.....} : prefijo **001 002 003 004**
- {.....}<postfijo> : postfijo **a.txt b.txt c.txt**
- <prefijo>{.....}<postfijo> : ambos **a001.txt a002.txt a003.txt**

# comandos: { } (brace expansion)

Bajar 6 archivos con wget (6 argumentos a wget):

```
[usuario@localhost ~]$ wget
http://example.com/doc/slides_part{1,2,3,4,5,6}.html
[usuario@localhost ~]$ wget http://example.com/doc/slides_part{1..6}.html


[usuario@localhost ~]$ mkdir {caso,datos,notas}

[usuario@localhost ~]$ echo 0{100..110}_A.txt

0100_A.txt 0101_A.txt 0102_A.txt 0103_A.txt 0104_A.txt 0105_A.txt
0106_A.txt 0107_A.txt 0108_A.txt 0109_A.txt 0110_A.txt
```

{001..100} 001 002 …010...099 100 : bash 4 admite “padding”

# comandos

```
$ echo {A..Z}{0..9}
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 C0 C1 C2 C3 C4 C5 C6
C7 C8 C9 D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 F0 F1 F2 F3
F4 F5 F6 F7 F8 F9 G0 G1 G2 G3 G4 G5 G6 G7 G8 G9 H0 H1 H2 H3 H4 H5 H6 H7 H8 H9 I0
I1 I2 I3 I4 I5 I6 I7 I8 I9 J0 J1 J2 J3 J4 J5 J6 J7 J8 J9 K0 K1 K2 K3 K4 K5 K6 K7
K8 K9 L0 L1 L2 L3 L4 L5 L6 L7 L8 L9 M0 M1 M2 M3 M4 M5 M6 M7 M8 M9 N0 N1 N2 N3 N4
N5 N6 N7 N8 N9 O0 O1 O2 O3 O4 O5 O6 O7 O8 O9 P0 P1 P2 P3 P4 P5 P6 P7 P8 P9 Q0 Q1
Q2 Q3 Q4 Q5 Q6 Q7 Q8 Q9 R0 R1 R2 R3 R4 R5 R6 R7 R8 R9 S0 S1 S2 S3 S4 S5 S6 S7 S8
S9 T0 T1 T2 T3 T4 T5 T6 T7 T8 T9 U0 U1 U2 U3 U4 U5 U6 U7 U8 U9 V0 V1 V2 V3 V4 V5
V6 V7 V8 V9 W0 W1 W2 W3 W4 W5 W6 W7 W8 W9 X0 X1 X2 X3 X4 X5 X6 X7 X8 X9 Y0 Y1 Y2
Y3 Y4 Y5 Y6 Y7 Y8 Y9 Z0 Z1 Z2 Z3 Z4 Z5 Z6 Z7 Z8 Z9
```

el comando **seq** también permite generar secuencias parametrizadas

# Source

Cuando se hace **source** de un script, éste se ejecuta y deja disponibles todas las variables que definió para quien lo invoca.

Se puede usar para compartir configuraciones comunes sin reescribirlas cada vez.

# comandos: “ . ” - source

```
[usuario@localhost ~]$ echo $TRABAJO

[usuario@localhost ~]$ cat variables
TRABAJO=/home/usuario/trabajo01
[usuario@localhost ~]$ . variables
[usuario@localhost ~]$ echo $TRABAJO
/home/usuario/trabajo01
```

# .bashrc

- Es un **script** que se ejecuta (source) cada vez que abrimos una terminal interactiva.
- Es un archivo oculto localizado en nuestro home (puede que no exista)
- **~/.bashrc**
- Cuando tenemos un **login shell** también se hace source de /etc/profile y ~/.bash_profile
- Es un buen lugar para definir alias y configurar variables de entorno personales.

# Scripts

Un **script** es un archivo de texto que agrupa comandos, ejecutarlo es equivalente a introducir su contenido en la consola línea por línea.

Para poder ejecutarlo debe tener **atributo de ejecución**.

También puede recibir **argumentos** para poder parametrizar los comandos que contiene.

# comandos

```
[usuario@localhost ~]$ cat script.sh
#!/bin/bash
#comentario
echo cantidad de argumentos: $#
echo Argumento 1: $1
echo Argumento 2: $2
[usuario@localhost ~]$ chmod +x script.sh
[usuario@localhost ~]$ ./script.sh arg1 "segundo arg"
cantidad de argumentos: 2
Argumento 1: arg1
Argumento 2: segundo arg
```

# Scripts: estructuras de control

Hay estructuras de control equivalentes a las de un lenguaje de programación estructurado, pero no las veremos en profundidad:

```
if   <comando> |  [ <condicion> ]  ; then
  <comandos>
fi
```

# scripts: estructuras de control

```
for  <variable> in <lista>; do
  <comandos que usan variable>
done

while <condicion> ; do
  <comandos>
done
```

# scripts: estructuras de control

```
case <expresion> in
    caso_1 )
      <comandos>;;
    caso_2 )
      <comandos>;;
    ......
esac
```

# Tutoriales de scripting recomendados

- http://wiki.bash-hackers.org/scripting/tutoriallist
- http://tldp.org/LDP/Bash-Beginners-Guide/html

Bash es recomendable para **automatización básica**, para tareas más complejas y código más fácil de mantener les sugiero aprender **Python**.

# Redirección

El comando no sabe realmente desde donde lee y hacia donde escribe.

La entrada y salida se **abstraen como archivos** que Bash se encarga de administrar. Si no se especifican los archivos se hace desde teclado y hacia la pantalla.

Esto permite que, independientemente del comando, podamos escribir la salida a un archivo, cambiar la entrada manual por una almacenada o descartar toda la salida, entre otras cosas.

# Redirección

Por convención se utilizan los siguientes **nombres** y **números** de descriptores de archivos:

- Entrada/stdin  (file descriptor 0)
- Salida/stdout  (file descriptor 1)
- Errores/stderr (file descriptor 2)

stdout es **buffered** y stderr es **unbuffered** (por si el  programa se muere antes de escribir)

# Redirección

Si vemos por ejemplo que archivos tiene abierta una consola bash al inicio observamos que la E/S está conectada a un emulador de terminal (pseudo consola).

stdin(0) -> /dev/pts/1
stdout(1) -> /dev/pts/1
stderr(2) -> /dev/pts/1

En el caso de una terminal real podría ser /dev/tty1

# Redirección

```
[usuario@localhost ~]$ lsof -ap $BASHPID
COMMAND  PID USER     FD   TYPE DEVICE SIZE/OFF NODE     NAME
bash     3809 usuario cwd  DIR  8,7      4096    2883586  /home/usuario
bash     3809 usuario rtd  DIR  8,7      4096    2        /
bash     3809 usuario txt  REG  8,7      960472  787154   /usr/bin/bash
bash     3809 usuario mem  REG  8,7      2127336 787079   /usr/lib64/libc-2.17.so
...
...
bash     3809 usuario 0u   CHR  136,1       0t0      4        /dev/pts/1
bash     3809 usuario 1u   CHR  136,1       0t0      4        /dev/pts/1
bash     3809 usuario 2u   CHR  136,1       0t0      4        /dev/pts/1
bash     3809 usuario 255u CHR  136,1       0t0      4        /dev/pts/1
```

# Redirección de output: n> archivo

```
$ echo hola >archivo
```

stdin(0) -> /dev/pts/1
stdout(1) -> archivo
stderr(2) -> /dev/pts/1

`comando >archivo` es equivalente a `comando 1>archivo`

# Redirección de output: n> archivo

Del mismo modo:

`comando 2>archivo` redirecciona los **errores** al archivo.

Otros números crean file descriptors nuevos, que en general no hacen nada o podían usarse dentro de un script.

**comando>archivo** **sobreescribe** o crea el archivo
**comando>>archivo** crea o **agrega** al archivo

# Redirección

La redirección se hace **antes** de ejecutar el comando por lo que si redireccionamos **al mismo archivo** con la intención que el comando lea, modifique y escriba no obtendremos el resultado esperado.

```
$ sort archivo > archivo
```

<--NO!

Primero redirecciona la salida a archivo y lo trunca, luego llama a sort sobre un archivo vacío y no escribe nada en archivo.

# Redirección

El comando **tee** permite redireccionar a más de un archivo a la vez.

Lo podríamos usar para redireccionar a un archivo y al mismo tiempo ver en pantalla la salida.

# Redirección de input: n< archivo

```
$ head <archivo
```

stdin(0) -> archivo
stdout(1) -> /dev/pts/1
stderr(2) -> /dev/pts/1

`comando <archivo` es equivalente a `comando 0>archivo`

Para varios comandos es redundante dado que pueden tomar como primer parámetro un archivo al que reemplazan por stdin.

# Redirección (duplicación): n>archivo **m>&n**

Podríamos querer redireccionar stdout y stderr al mismo archivo, Para esto contamos con el atajo &n que indica al mismo lugar que se redireccionó n.

```
$ comando >salida 2>salida
$ comando >salida 2>&1
$ comando_molesto >/dev/null 2>&1
```

# Background

Cuando un proceso corre en **foreground** hay que esperar a que termine (o interrumpirlo) para ejecutar el siguiente comando.

También es posible enviar al **background** un comando apenas lo escribimos o en medio de la ejecución  y continuar ingresando comandos.

# comandos: &, <ctrl-z>, jobs, fg n

```
[usuario@localhost ~]$ cp -r carpeta carpeta_copia&
[1] 7945
[usuario@localhost ~]$ vi nada
<ctrl-z>
[2]+  Detenido                 vim nada
[usuario@localhost ~]$ jobs
[1]-  Ejecutando               cp -r carpeta carpeta_copia &
[2]+  Detenido                 vim nada
[usuario@localhost ~]$ fg 2
```

# Background

Otras opciones para correr en background son los comandos:

- **nohup** : automáticamente redirecciona la salida y envía al background al mismo tiempo.
- **screen :** permite mantener varias sesiones interactivas al mismo tiempo o desconectarse y retomar luego (manejador de ventanas para consola)

# Pipes |

Un pipe es lo que permite **combinar** los comandos. Crea entre otras cosas un archivo especial al que se redirecciona la salida del primer comando y la entrada del segundo. Se pueden encadenar tantos como se necesiten.

```
[usuario@localhost ~]$ grep caso1 archivo | wc -l
20
[usuario@localhost ~]$ grep caso1 archivo | grep -v rasgo2 | wc -l
5
```

# Varios comandos

Parte de la filosofía UNIX es tener comandos con **una tarea** específica y que se puedan **combinar**.

A continuación veremos la tarea principal de varios comandos.

Esto les permitirá reconocer que comando deben usar para cada tarea y buscar por su cuenta los flags y parámetros necesarios para el resultado que necesiten.

# Recordemos cómo obtener ayuda

- `$<comando> --help`
- `$<comando> -h` (si -h no se usa para otra cosa)
- `$help <comando>` (para bultins)
- `$man <comando>`
- `$man -k <palabra clave>`
- `$apropos <palabra clave>`

- explainshell
- stackoverflow

# Comandos útiles

- echo, printf
- grep
- sort
- uniq
- wc
- nl
- tr
- cut
- sed
- awk
- paste
- diff
- yes
- xargs
- bc
- date
- watch
- tail -f

# Mostrar texto: echo, printf

Para textos sencillos o valores de variables: echo

```
$ echo “el valor de la variable es $VARIABLE”
```

Para texto formateado, decimales, padding: printf

```
$ printf "%10.5f\n" 17.2
 17.20000
```

También puede guardar en una variable el string:

```
$ printf -v SALUDO "Hola %s" "$LOGNAME"
```

# Filtrar: head, tail

Permite ver sólo la primera o la última parte de la entrada, por defecto son 10 lineas

- -n <numero> ver <numero> de lineas

```
$ head -3 archivo.txt
linea1
linea2
linea3
```

para paginar una salida extensa usar: less

# Filtrar: grep (global regular expression print)

Puede buscar dentro de un archivo o en todos los archivos de la carpeta, o para filtrar una salida al usarlo con un pipe.
Tiene muchas opciones para patrones y formas de mostrar las coincidencias

```
$ grep usuario /etc/passwd
$ ps aux | grep firefox
```

- -v invertir la búsqueda
- -i ignorar mayúsculas/minúsculas

# Ordenar: sort

Ordenar un archivo o salida: sort

- -k tomar como clave una columna o un rango de caracteres
- -n orden numérico / -h orden “humano (K,M,G)”
- -t usar otro separador de columnas en vez que la transición blanco-noblanco

```
$ ls -lh /home/$USER | sort -h -k5
```

# Filtrar: uniq

Remover repetidos, deben estar juntos por lo que en general se utiliza en combinación con sort

- -c cuenta ocurrencias

```
$ sort casos.txt | uniq -c
```

# Contar: wc (word count)

Cuenta palabras, líneas y caracteres, por defecto muestra las tres estadisticas.

- -l cuenta lineas

```
$ wc /usr/share/dict/words
 479828  479828 4953680 /usr/share/dict/words
```

```
$ grep caso1 casos.txt | wc -l
(se podría hacer con grep -c)
```

# Numerar: nl (number lines)

Agrega a la entrada el número de línea de cada línea

- -s separar el número del original con un carácter
- -l unir varias líneas en blanco en una

```
$ head -5 /usr/share/dict/words | nl
     1    1080
     2    10-point
     3    10th
     4    11-point
     5    12-point
```

# Reemplazar: tr (transliterate)

Reemplazar un conjunto de **caracteres** por otro.

- -d borra, no reemplaza

```
$ echo aabbcc | tr ab 12
1122cc
$ salida_con_espacios | tr " " "\n"
palabra1
palabra2
palabra3
$ echo hola | tr [a-z] [A-Z]
HOLA
```

# Filtrar: cut

Muestra sólo algunas columnas de una salida

- -d establecer delimitador
- -f columna o rango de columnas a mostrar (fields)
- -c rango de carateres, no columnas

```
$grep "Módulo 2" inscripcion.csv | cut -d, -f3
$ cut -c10..13 archivo
ABC
XYZ
```

# Editar: sed (stream editor)

Es un completo editor que toma **comandos** internos a realizar sobre una entrada. Soporta expresiones regulares para patrones complejos.

Un uso sencillo es reemplazar cadenas de texto:

```
$ sed 's/alpha/beta/' archivo1 > archivo2
```

http://sed.sourceforge.net/grabbag/tutorials/do_it_with_sed.txt

# Manipular texto: awk (Aho, Weinberger y Kernighan )

Es un completo lenguaje de programación excelente para manipular texto tabulado, con encabezados, separadores y aritmética incorporada.

```
$ awk -F: '{print $1}' /etc/passwd
$ awk '{if ($2 >100) print $3}' datos
```

(funcionalidad cut)

http://www.grymoire.com/Unix/Awk.html

# Juntar líneas: paste

Permite juntar línea por línea dos o más archivos

```
$ paste serie1 serie2 > serie12
```

# Buscar diferencias: diff

Permite comparar archivos y resaltar las diferencias o mostrar solo las líneas iguales.

- -y mostrar las líneas una al lado de la otra

```
$ diff archivo_v1 archivo_v2
```

# Pajarito: yes

Produce tantos yes como sean necesarios. Se usa en combinación con un pipe o redirección. En general los comandos que requieren confirmación proveen un flag -y o -f con el mismo objetivo

- yes <texto> produce infinitos <texto>

```
$ yes | fsck /dev/sda2
```

# Calculadora: bc (basic calculator)

Calculadora de precisión arbitraria, toma el texto que se ingresa, lo interpreta como una expresión aritmética y calcula su resultado.

- scale indica la precisión

```
$ echo "scale=6;  60/7.02" | bc
8.547008
```

# Fecha: date

Devuelve la fecha actual y permite hacer aritmética de fecha. Tiene modificadores para presentarla con múltiples formatos (día de la semana, número de semana del año, etc)

- -d otra fecha distinta de la actual

```
$date -d"next monday"
lun may  7 00:00:00 -03 2018
$ date -d “apr 26+28days”
```

# Monitorear comando: watch

permite ejecutar periódicamente un comando para evitar el combo flecha_arriba+enter indefinidamente

- -n intervalo en segundos

```
$watch free
```

# Monitorear archivo: tail -f

muestra el final de un archivo que se está escribiendo y se actualiza con cada escritura. Se interrumpe con <ctrl-c>

```
$ tail -f salida_programa

# tail -f /var/log/secure | grep root
```

# Links

- Tutorial abarcativo sobre scripts:
  - http://tldp.org/LDP/Bash-Beginners-Guide/html
- Alias/hacks:
  - https://wiki.archlinux.org/index.php/Bash
- Todo sobre redirección <>:
  - http://wiki.bash-hackers.org/howto/redirection_tutorial
- Screen:
  - https://www.linode.com/docs/networking/ssh/using-gnu-screen-to-manage-persistent-terminal-sessions/
  - https://www.rackaid.com/blog/linux-screen-tutorial-and-how-to