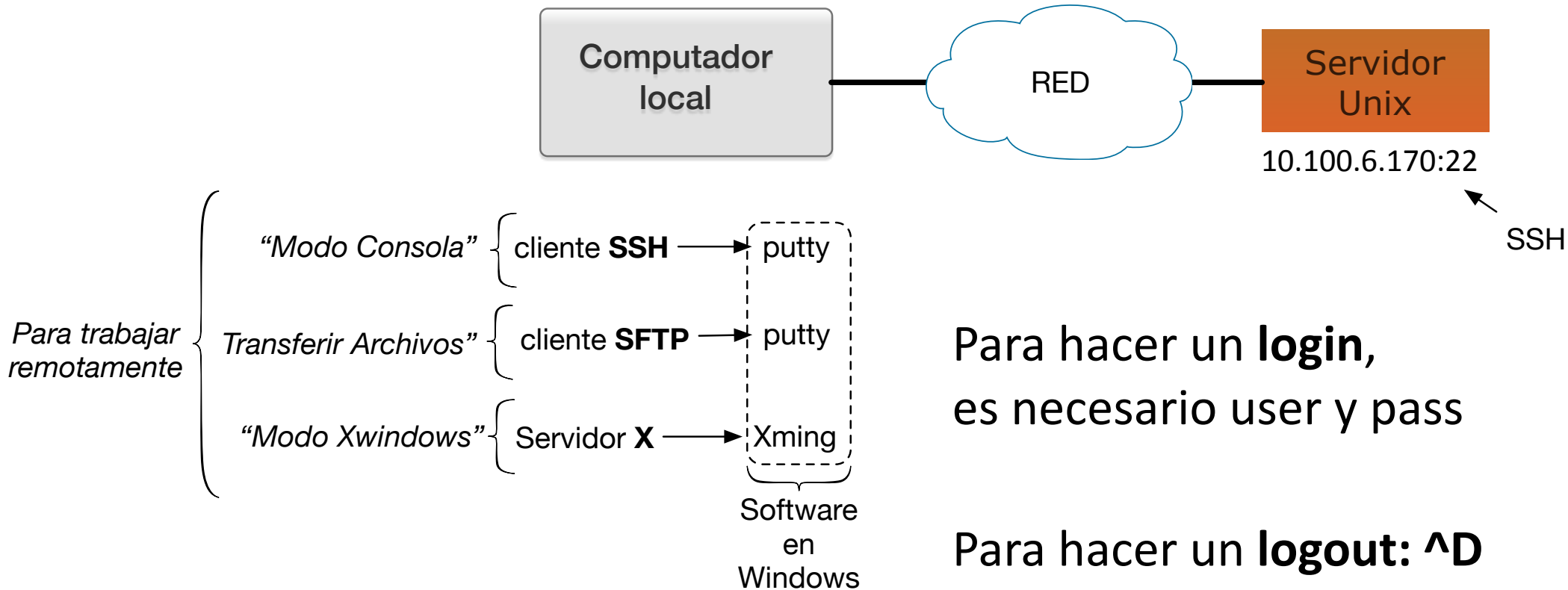


Unix básico y programación Shell

Forma de trabajo

Esquema de trabajo



Unix Básico

Capítulos del Libro “The Linux(R) Command Line”

Descripción General del shell,
manipulación de Archivos y
Directorios, Comandos Básicos,

1 al 5, 7, 8

Permisos de Archivos y Directorios

9

Configuración del Shell

11, 13

Control de Procesos, Señales

10

Programación en Shell (Scripting)

24 al 36

Descripción general del shell BASH

Capítulo 1 “The Linux command Line”

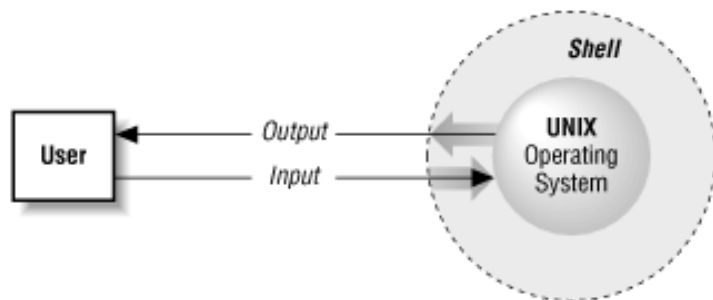
Descripción general

Un “shell es un programa que interpreta comandos ingresados y puede ejecutar otros programas.

Normalmente tienen un lenguaje de programación que tiene el mismo nombre del shell.

Es la forma más sencilla de interactuar con el Sistema Operativo

Se accede localmente a través del terminal real del computador, emuladores de terminal gráficos o a través de conexiones remotas como SSH.



Shell instalados en un sistema:

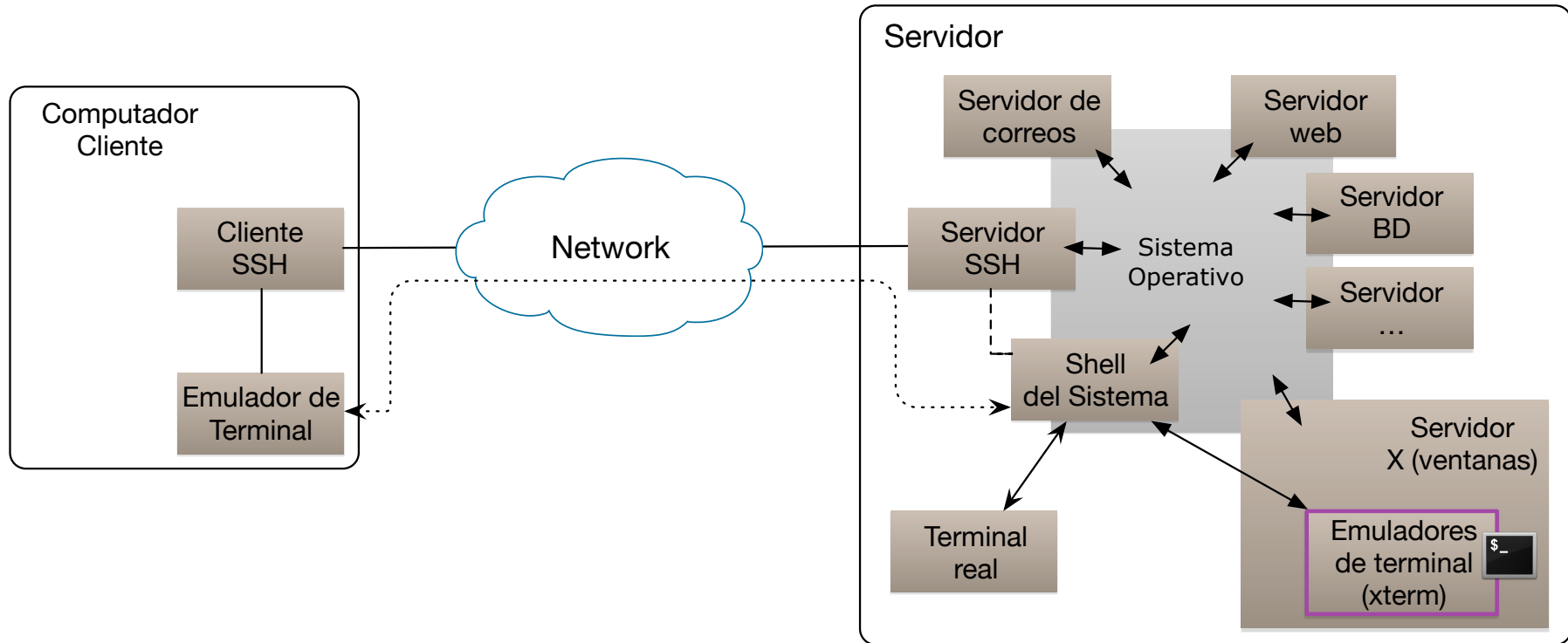
Hay varios tipos

csh, tcsh, ksh, zsh, bash

/etc/shells



Diagrama general



```
gabriel — gabriel@server01: ~ — ssh gabriel@10.100.6.166 — 78x19
Last login: Thu Jan  7 11:12:54 on ttys002
Gabriels-iMac:~ gabriel$ ssh gabriel@10.100.6.166
gabriel@10.100.6.166's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Jan  6 17:11:47 2016 from 10.100.6.175
gabriel@server01:~$
```

Computador local
(comando ssh equivale al cliente SSH)

Cliente SSH se conecta al servidor SSH requerido, y éste procede a **autenticar** al usuario que se conecta con la **base de datos de usuarios del sistema**.

El usuario se valida y el servidor SSH **llama al shell del usuario**, el que imprime un mensaje, **prepara el ambiente de trabajo** y **queda a la espera de comandos**.

Se acceden a través del comando **man**

Se dividen en secciones:

- 1 Executable programs or shell commands
- 2 System calls (functions provided by the kernel)
- 3 Library calls (functions within program libraries)
- 4 Special files (usually found in /dev)
- 5 File formats and conventions eg /etc/passwd
- 6 Games
- 7 Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
- 8 System administration commands (usually only for root)
- 9 Kernel routines [Non standard]

Convención:

“[...] para cumplir con los requerimientos del desarrollo, deberá utilizar la función **printf(3)**, [...]”

man printf

```
PRINTF(1)                                User Commands                                PRINTF(1)

NAME
    printf - format and print data

SYNOPSIS
    printf FORMAT [ARGUMENT]...
    printf OPTION

DESCRIPTION
    Print ARGUMENT(s) according to FORMAT, or execute according to OPTION:

    --help display this help and exit

    --version
        output version information and exit

    FORMAT controls the output as in C printf.  Interpreted sequences are:

    \"    double quote
    \\    backslash
```

man 3 printf

```
PRINTF(3)                                Linux Programmer's Manual                                PRINTF(3)

NAME
    printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf - formatted
    output conversion

SYNOPSIS
    #include <stdio.h>

    int printf(const char *format, ...);
    int fprintf(FILE *stream, const char *format, ...);
    int sprintf(char *str, const char *format, ...);
    int snprintf(char *str, size_t size, const char *format, ...);

    #include <stdarg.h>

    int vprintf(const char *format, va_list ap);
    int vfprintf(FILE *stream, const char *format, va_list ap);
    int vsprintf(char *str, const char *format, va_list ap);
    int vsnprintf(char *str, size_t size, const char *format, va_list ap);

    Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

    snprintf(), vsnprintf():
        _BSD_SOURCE || _XOPEN_SOURCE >= 500 || _ISOC99_SOURCE || _POSIX_C_SOURCE >= 200112L;
        or cc -std=c99
```

Comandos básicos

- `man`
- `ls`, `mkdir`, `cp`, `rm`, `mv`, `ln`
- `touch`, `file`, `diff`
- `cat`, `more`, `less`, `tail`, `head`
- `tar`, `gzip`, `bzip2`
- `grep`, `cut`, `col`
- `alias`
- `df`, `du`
- `ps`, `top`, `renice`, `kill`, `fg`, `bg`
- `crontab`, `at`

Revisar páginas de manual
de cada comando

ls, rm Listar, remover archivos.

mkdir, rmdir Crear, remover directorios.

ln Crea un link simbólico.

touch Crear, actualizar hora de archivos.

cat, more, less Visualiza contenido de archivos.

file Determina el tipo de archivo.

tail, head Extrae el final y el comienzo de un archivo.

gzip, bzip2 Comprimir archivo.

tar Agrupar archivos en uno sólo.

df Muestra el espacio ocupado por los sistema de
archivos.

du Lo mismo, pero por archivos.

grep Muestra las líneas que coinciden con cierto
patrón.

```
grep 'sacar estas lineas' documento.txt
```

```
grep -v 'no mostrar esto' documento.texto
```

awk Lenguaje utilizado para extraer patrones en un
texto.

sort Ordena las líneas de un archivo.

find Busca archivos.

```
find / -name "*.c" -print
```

```
find / -name "*.bak" -exec rm -r {}
```

Información del usuario

Los datos de los usuarios están en el archivo `/etc/passwd`

Sólo el root
puede modificarlo

Pero cada usuario puede modificar “su línea”

Comandos asociados

```
chsh (1)  
chfn (1)  
passwd (1)
```

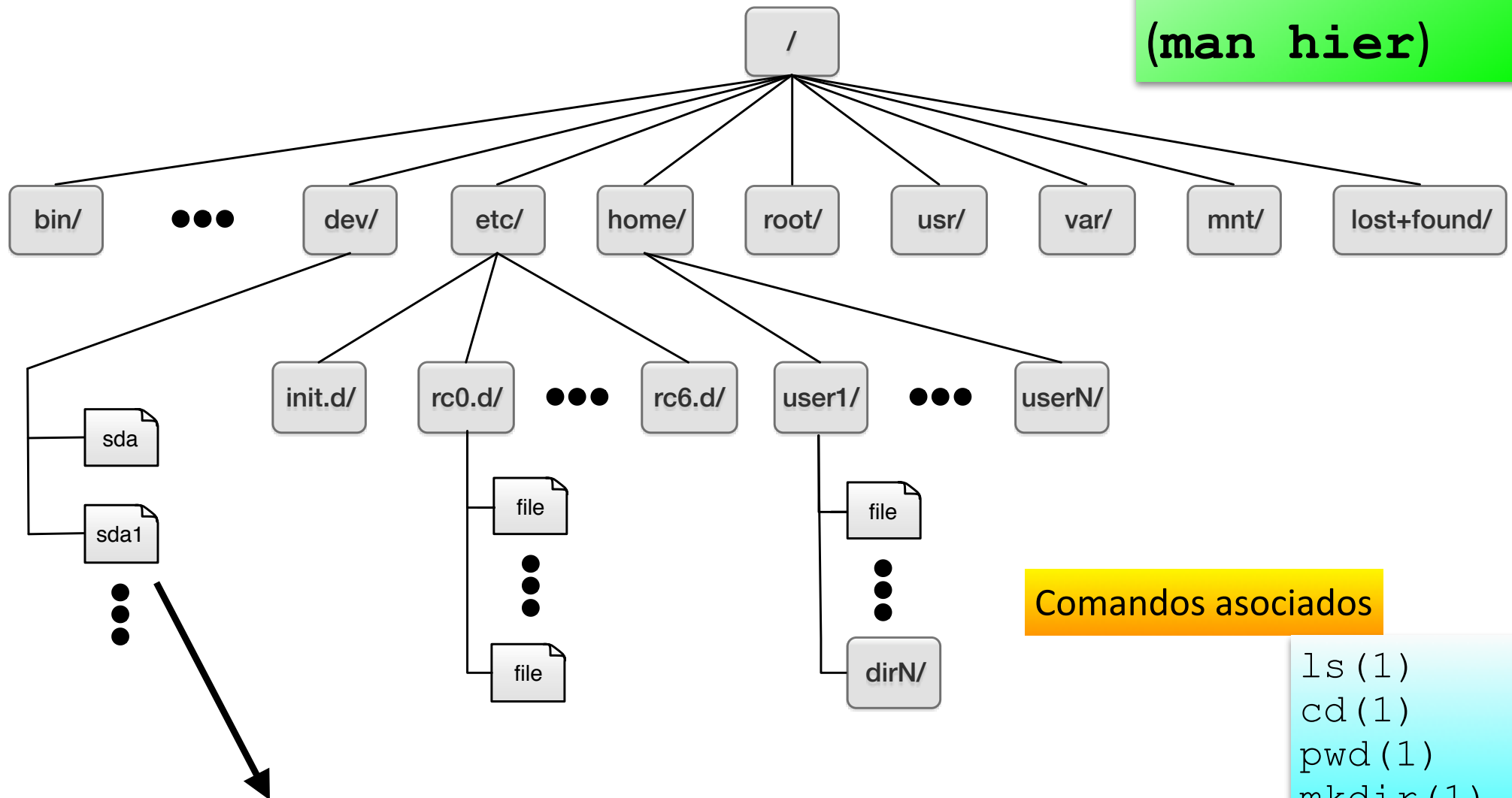
Los usuarios pueden visualizar los datos públicos de otros usuarios a través del comando **finger (1)**.

Archivos en Unix

Capítulo 2 y 3 “The Linux command Line”

Estructura del FS en Unix

Cada directorio tiene una función establecida
(**man hier**)

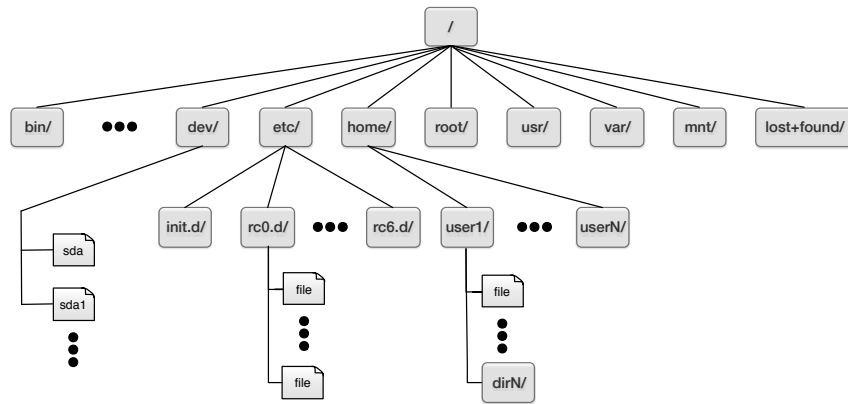


Comandos asociados

```
ls(1)
cd(1)
pwd(1)
mkdir(1)
rmdir(1)
rm(1)
```

Todo dispositivo de HW está mapeado a un archivo

Línea de comandos



Los comandos son programas que están localizadas en ciertos directorios

`/bin, /sbin/, /usr/bin, /usr/sbin`

El SHELL tiene una variable de entorno (**PATH**) que controla en qué directorios debe buscar los comandos que se ingresan

```
[gabriel@server01:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/home/gabriel/bin
```

Consideraciones

El directorio **home** del usuario **jperez** es **~jperez**

El directorio **home** de uno mismo del **~**

Los directorios se separan con el caracter **/**

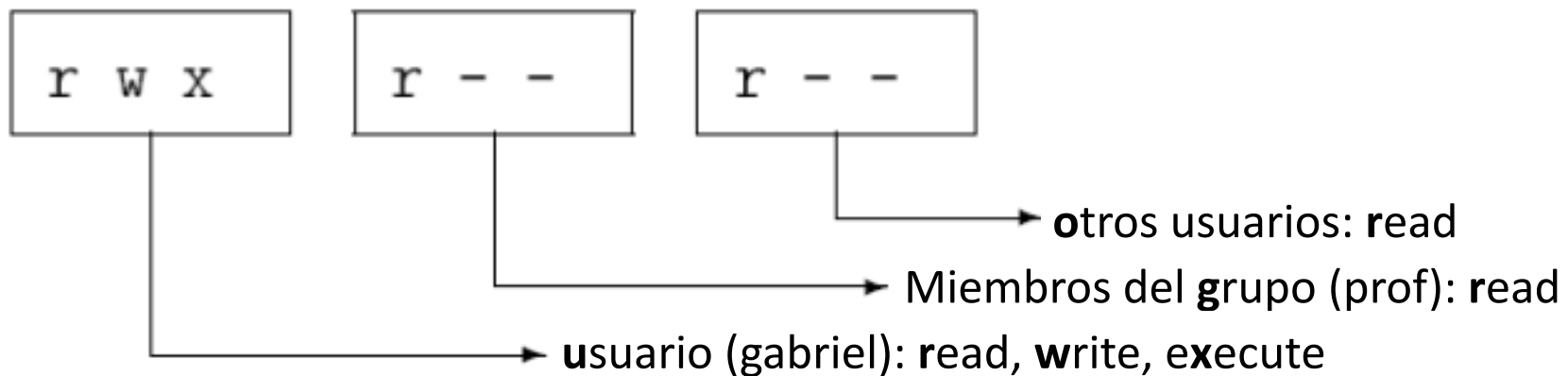
```
[gabriel@server01:~/public_html$ ls -l
total 8
drwxr-xr-x 2 gabriel prof 4096 Dec 21 15:24 C
drwxr-xr-x 2 gabriel prof 4096 Dec 21 15:26 perl
[gabriel@server01:~/public_html$ pwd
/home/gabriel/public_html
[gabriel@server01:~/public_html$ cd perl/
[gabriel@server01:~/public_html/perl$ ls -l
total 8
-rwxr-xr-x 1 gabriel prof 105 Dec 21 15:26 test.pl
-rwxr-xr-x 1 gabriel prof 94 Dec 21 15:25 test.pl~
[gabriel@server01:~/public_html/perl$ cd /usr/lib/X11/
[gabriel@server01:/usr/lib/X11$ pwd
/usr/lib/X11
[gabriel@server01:/usr/lib/X11$ cd ~
[gabriel@server01:~$ pwd
/home/gabriel
[gabriel@server01:~$ cd /usr/lib/X11/
[gabriel@server01:/usr/lib/X11$ cd
[gabriel@server01:~$ pwd
/home/gabriel
gabriel@server01:~$
```

```
[test@server01:~$ id
uid=1001(test) gid=1003(alm) groups=1003(alm)
[test@server01:~$ ls -l
total 0
[test@server01:~$ cd ~gabriel
[test@server01:/home/gabriel$ ls -l
total 8
-rw-r--r-- 1 gabriel prof 73 Dec 30 15:45 holamundo.c
drwxr-xr-x 4 gabriel prof 4096 Dec 18 16:30 public_html
[test@server01:/home/gabriel$ rm holamundo.c
[rm: remove write-protected regular file 'holamundo.c'? yes
rm: cannot remove 'holamundo.c': Permission denied
test@server01:/home/gabriel$
```

Unix tiene un sistema de permisos de archivos “decente”

Permisos de archivos

```
gabriel@server01:~$ ls -l
total 8
-rw-r--r-- 1 gabriel prof 73 Dec 30 15:45 holamundo.c
drwxr-xr-x 4 gabriel prof 4096 Dec 18 16:30 public_html
```



3 categorías de permisos: read, **w**rite, **x**ecute

3 categorías de usuarios: **u**ser, **g**roup, **o**ther

Comandos asociados

```
chmod (1)
chgrp (1)
chown (1)
```

Configuración del Shell

Son variables que el shell puede utilizar. Su valor se accede anteponiendo el signo **\$** antes del nombre de la variable.

Se pueden conocer a través del comando **env**

PATH: Controla qué directorios y en que orden el shell busca comandos.

_: Último comando ejecutado

```
[gabriel@server01:~$ pwd
/home/gabriel
[gabriel@server01:~$ echo $_
pwd
[gabriel@server01:~$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
[gabriel@server01:~$ echo $USER
gabriel
[gabriel@server01:~$ echo $SSH_CONNECTION
10.100.6.175 51869 10.100.6.166 22
```

Se pueden crear nuevas variables

```
[gabriel@server01:~$ A=10
[gabriel@server01:~$ echo $A
10
[gabriel@server01:~$ fecha=$(date +%Y-%m-%d)
[gabriel@server01:~$ echo $fecha
2016-01-06
[gabriel@server01:~$
```

Cuando se realiza un login al sistema, BASH ejecuta el archivo `~/ .bashrc`

Primeras líneas del archivo `.bashrc`

```
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples

# If not running interactively, don't do anything
case $- in
    *i*) ;;
    *) return;;
esac

# don't put duplicate lines or lines starting with space in the history.
# See bash(1) for more options
HISTCONTROL=ignoreboth

# append to the history file, don't overwrite it
shopt -s histappend

# for setting history length see HISTSIZE and HISTFILESIZE in bash(1)
HISTSIZE=1000
HISTFILESIZE=2000

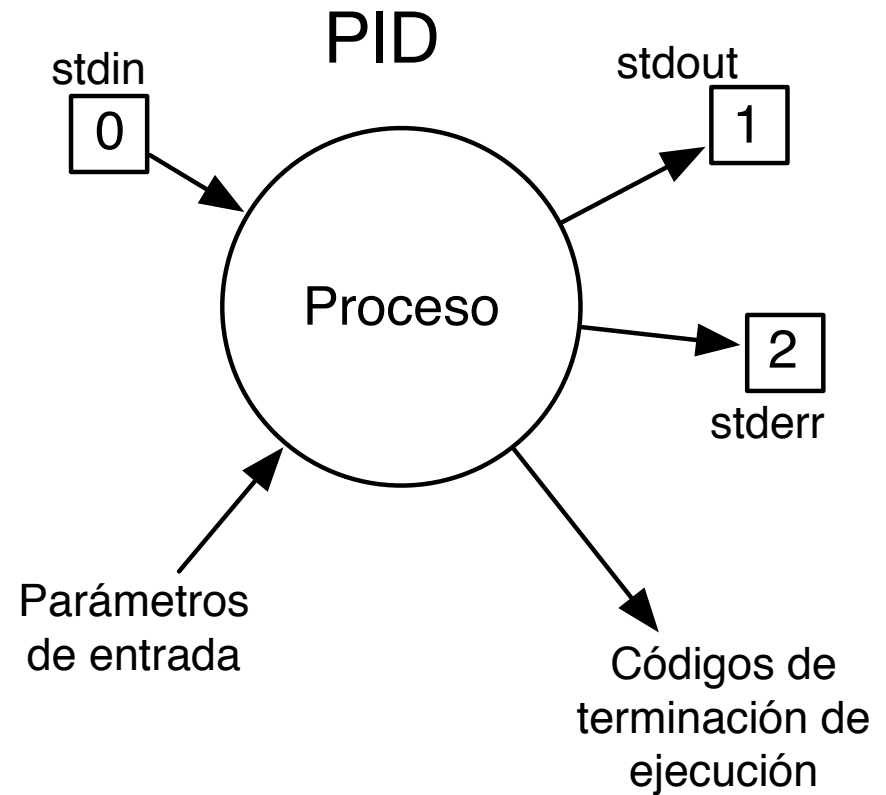
# check the window size after each command and, if necessary,
# update the values of LINES and COLUMNS.
shopt -s checkwinsize

# If set, the pattern "*" used in a pathname expansion context will
# match all files and zero or more directories and subdirectories.
#shopt -s globstar
```

Procesos (Sistemas Operativos)

Es todo programa que se está ejecutando en el sistema operativo
Se identifican con su PID

Standar Input
Standar Output
Standar Error
Parámetros de entrada
Códigos de término



Procesos en Shell

Gestión

procesos

gestión de procesos (recordar estados de un proceso)

Comandos

- `ps, top`
- `kill`
- `&, fg, bg`
- `nice`
- `CTRL+C, CTRL+Z`

kill -ID_SEÑAL PID Envía *ID_SEÑAL* al proceso *PID*
ID_SEÑAL KILL, HUP, CONT, TERM, STOP

programa & Ejecución en segundo plano.

& Permite que el programa programa se ejecute en “background”

CTRL+C Detiene el proceso y lo finaliza.

CTRL+Z Suspende el proceso actual y queda en la lista de tareas pendientes.

fg Reinicia el ultimo proceso suspendido y lo trae a primer plano.

bg Lo mismo, pero lo ejecuta en segundo plano “background”

ps Despliega información de procesos.

top Similar a ps, pero interactivo.

nice Permite ejecutar un proceso con una prioridad distinta a la por omisión.

renice Permite cambiar la prioridad de un proceso en ejecución.

top

```
last pid: 90392; load averages: 0.01, 0.09, 0.05 up 2+18:56:25 10:49:58
61 processes: 1 running, 60 sleeping
CPU states: 0.2% user, 0.0% nice, 0.2% system, 0.2% interrupt, 99.4% idle
Mem: 482M Active, 662M Inact, 221M Wired, 76M Cache, 163M Buf, 64M Free
Swap: 2032M Total, 100K Used, 2032M Free

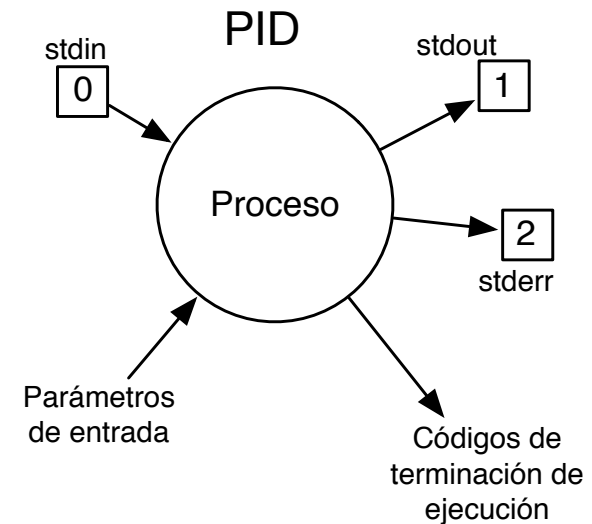
PID USERNAME PRI NICE SIZE RES STATE C TIME WCPU CPU COMMAND
90389 gabriel 28 0 2020K 1164K CPU0 0 0:00 1.36% 0.78% top
218 mysql 2 0 31996K 16020K poll 1 97:27 0.24% 0.24% mysqld
255 web 18 0 22592K 19900K lockf 1 2:10 0.15% 0.15% httpd
223 web 18 0 23768K 21108K lockf 1 2:38 0.00% 0.00% httpd
256 web 2 0 23540K 20876K select 1 2:36 0.00% 0.00% httpd
233 web 18 0 24016K 21372K lockf 1 2:35 0.00% 0.00% httpd
176 www 18 0 108M 25160K pause 0 2:29 0.00% 0.00% java
229 web 18 0 23424K 20736K lockf 1 2:26 0.00% 0.00% httpd
232 web 18 0 23376K 20728K lockf 1 2:25 0.00% 0.00% httpd
254 web 2 0 23772K 21032K sbwait 1 2:25 0.00% 0.00% httpd
220 web 2 0 24388K 21732K sbwait 1 2:24 0.00% 0.00% httpd
```


Procesos en Shell

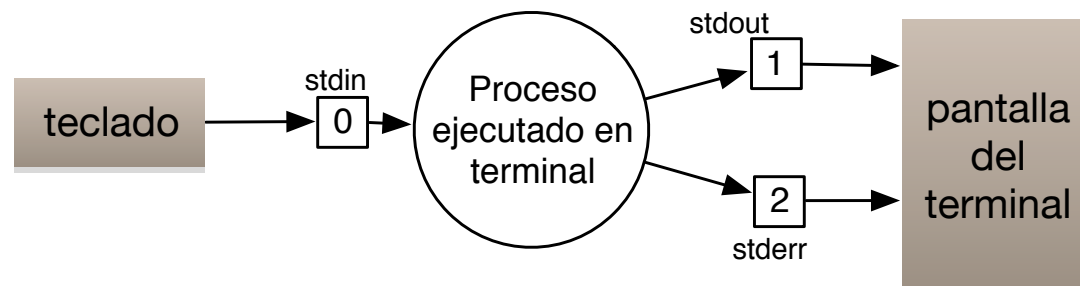
Comunicación entre procesos

Entrada y Salidas de un proceso

Standar Input
Standar Output
Standar Error
Parámetros de entrada
Códigos de término



Normalmente, un proceso que se ejecuta en el terminal tiene la siguiente asignación de entrada y salidas



Pipes (| , | &)

Permite conectar el `stdout` de un proceso con el `stdin` de otro

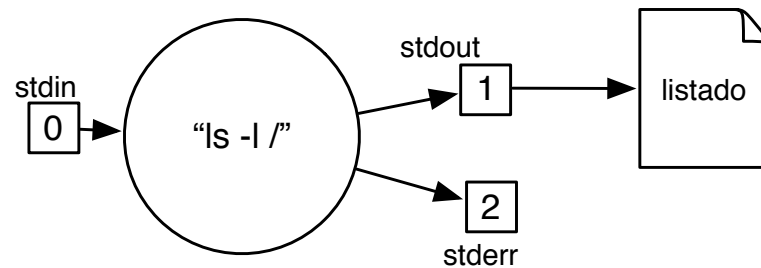
Redireccionamiento (>, >>, <, >&)

Permite que una salida de un proceso sea almacenada en un archivo o que el contenido de un archivo sea interpretado como el `stdin` del proceso¹.

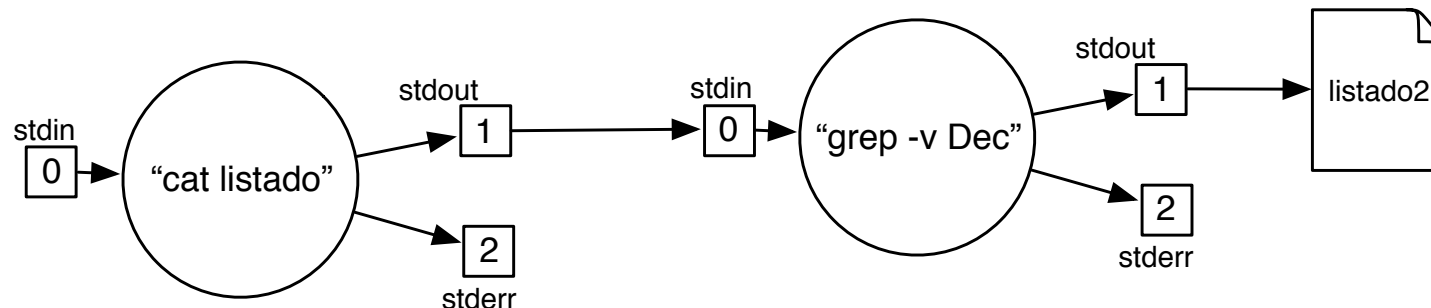
¹<http://www.gnu.org/software/bash/manual/bash.html#Redirections>

Comunicación entre procesos

Almacenar, en el directorio actual, la información (permisos, dueño, fecha de creación) de cada subdirectorios del directorio / en un archivo denominado `listado`.

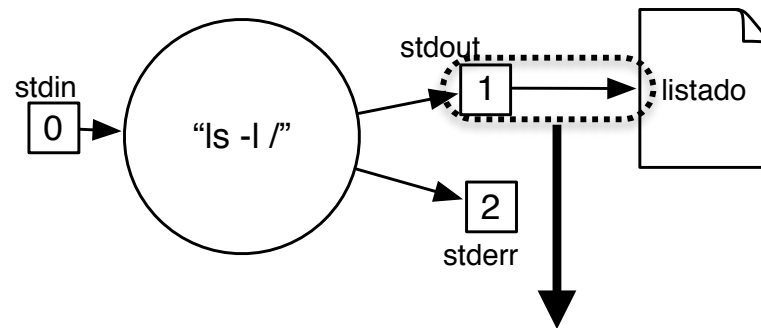


De dicho archivo, eliminar todos los subdirectorios que fueron creado en el mes de Diciembre, y los datos resultantes, guardarlos en en archivo `listado2`.



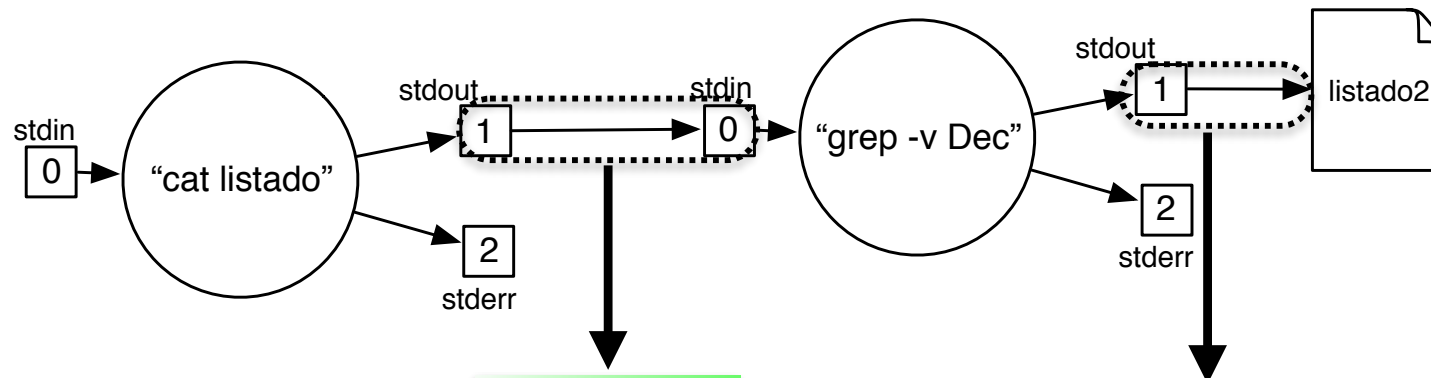
Comunicación entre procesos

Almacenar, en el directorio actual, la información (permisos, dueño, fecha de creación) de cada subdirectororio del directorio / en un archivo denominado `listado`.



Redirección desde stdout hacia un archivo

De dicho archivo, eliminar todos los subdirectorios que fueron creado en el mes de Diciembre, y los datos resultantes, guardarlos en en archivo `listado2`.

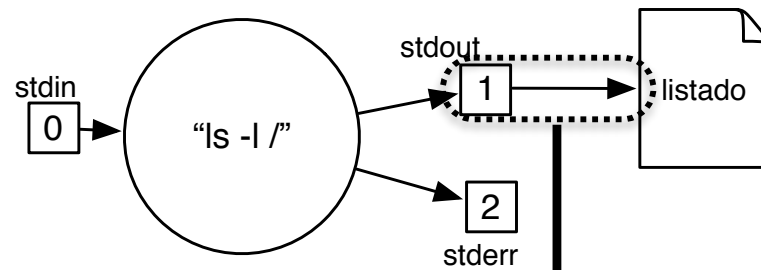


Conexión
salida-entrada

Redirección desde stdout hacia un archivo

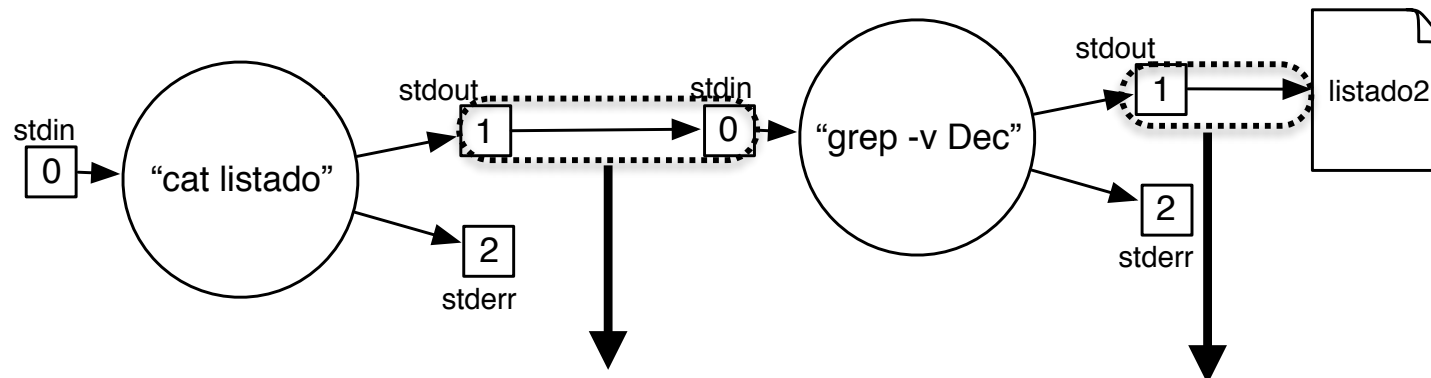
Comunicación entre procesos

Almacenar, en el directorio actual, la información (permisos, dueño, fecha de creación) de cada subdirectororio del directorio / en un archivo denominado `listado`.



```
$ ls -l / > listado
```

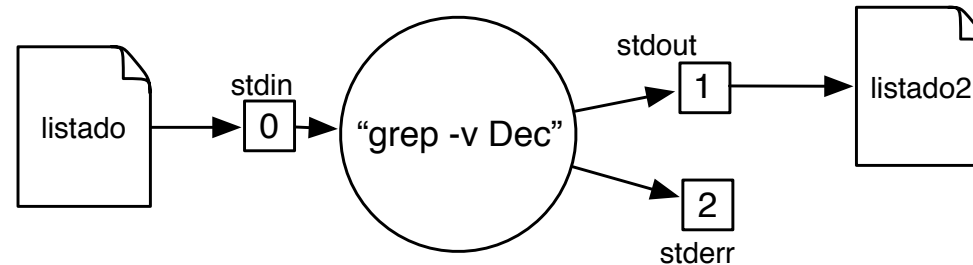
De dicho archivo, eliminar todos los subdirectorios que fueron creado en el mes de Diciembre, y los datos resultantes, guardarlos en en archivo `listado2`.



```
$ cat listado | grep -v Dec > listado 2
```

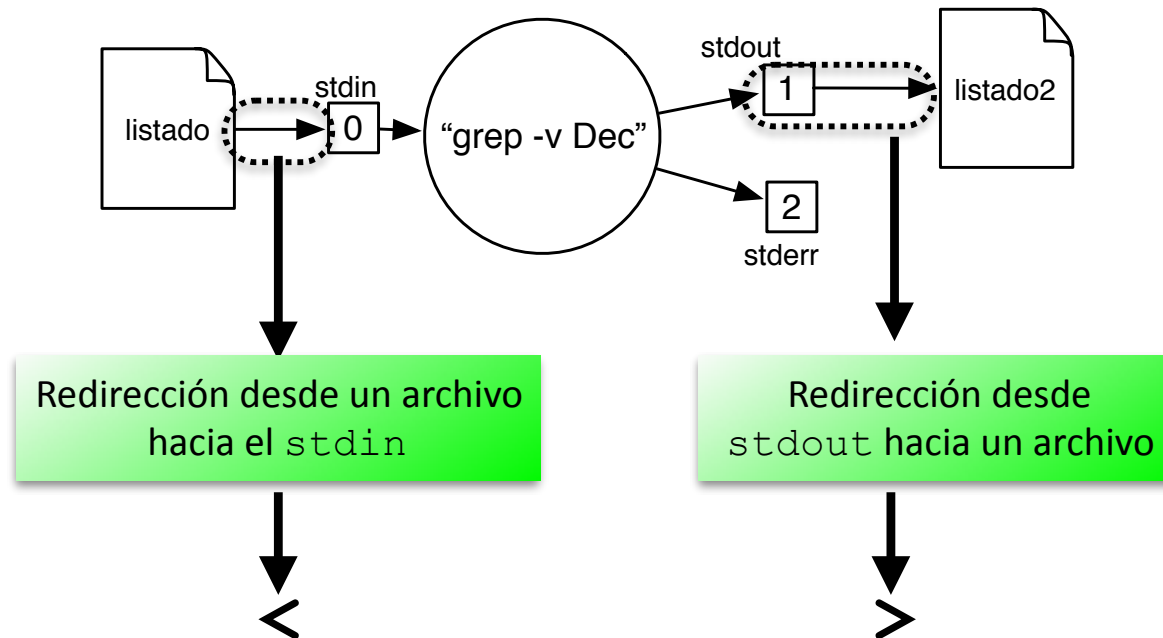
Comunicación entre procesos

De dicho archivo, eliminar todos los subdirectorios que fueron creado en el mes de Diciembre, y los datos resultantes, guardarlos en en archivo `listado2`.



Comunicación entre procesos

De dicho archivo, eliminar todos los subdirectorios que fueron creado en el mes de Diciembre, y los datos resultantes, guardarlos en en archivo `listado2`.



```
$ grep -v Dec < listado > listado 2
```


El comando `du -h /`, muestra en pantalla el tamaño de cada subdirectorio del directorio/. Si este comando se ejecuta como usuario, entregará líneas de error, que informan que faltan permisos para terminar la tarea.

Objetivo: guardar en un archivo (error.txt) las líneas de error y en otro archivo (size.txt) las líneas válidas.

Pista: el comando `>` tiene un operador a la izquierda, que es el descriptor de la salida que se quiere redireccionar

Ejemplo

```
ls -l / > listado
```

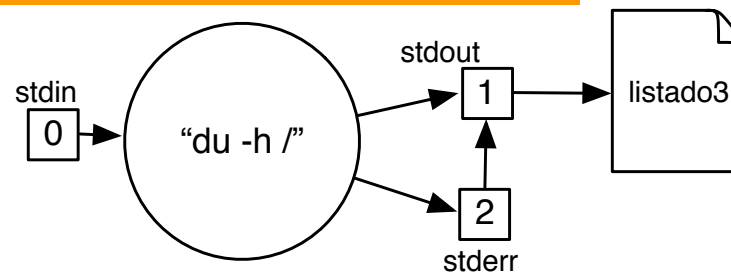
equivale
a

```
ls -l / 1> listado
```

El comando **du -h /**, muestra en pantalla el tamaño de cada subdirectorio del directorio/. Si este comando se ejecuta como usuario, entregará líneas de error, que informan que faltan permisos para terminar la tarea.

Objetivo: guardar el `stdout` y `stderr` en el mismo archivo, con una sólo operación de redirección

Paso 1: dibujar el diagrama de la solución



Paso 2: encontrar el comando de redirección que permita hacer lo diseñado

```
$ du -h / > listado3 2>&1
```

¿dónde está en el diagrama los operadores `>&` y `>`?

El comando `du -h /`, muestra en pantalla el tamaño de cada subdirectorio del directorio/. Si este comando se ejecuta como usuario, entregará líneas de error, que informan que faltan permisos para terminar la tarea.

Objetivo: Contar el total de líneas que arroja el comando, incluyendo las líneas de stdout y stderr. El comando que permite contar líneas desde su `stdin` es `wc -l`

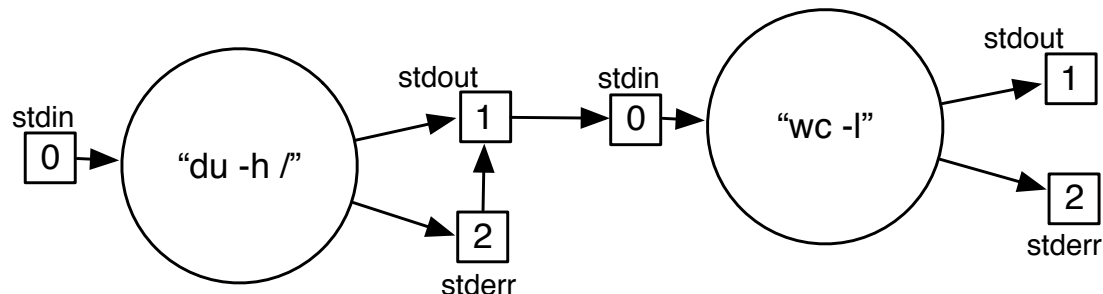
Paso 1: dibujar el diagrama de la solución

Paso 2: encontrar el comando de redirección que permita hacer lo diseñado

El comando **du -h /**, muestra en pantalla el tamaño de cada subdirectorio del directorio/. Si este comando se ejecuta como usuario, entregará líneas de error, que informan que faltan permisos para terminar la tarea.

Objetivo: Contar el total de líneas que arroja el comando, incluyendo las líneas de stdout y stderr

Paso 1: dibujar el diagrama de la solución



Paso 2: encontrar el comando de redirección que permita hacer lo diseñado

```
$ du -h / | & wc -l
```

¿dónde está en el diagrama el comando **| &**?

Comunicación entre procesos

Objetivo: Obtener la cantidad de datos (en Bytes) que ingresan/salen por la interfaz `eth0` de un computador y mostrarlos como `"bytes_in:bytes_out"`

1) Buscar dónde el SO entrega esa información

A través del comando `ifconfig(8)`

```
[gabriel@server01:~$ /sbin/ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 12:ed:fd:2e:7a:56
          inet addr:10.100.6.166  Bcast:10.100.6.255  Mask:255.255.255.0
          inet6 addr: fe80::10ed:fdff:fe2e:7a56/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:14569416 errors:0 dropped:0 overruns:0 frame:0
          TX packets:61765 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2128578844 (1.9 GiB)  TX bytes:8665348 (8.2 MiB)
```

2) Determinar cómo sacar esa información

```
[gabriel@server01:~$ /sbin/ifconfig eth0 |grep bytes|awk '{print $2":"$6}'| cut -d: -f 2,4
2128718843:8683368
```

3) Pruebe lo siguiente, ¿para que podría utilizarse?

```
$ net_count=$(/sbin/ifconfig eth0 |grep bytes|awk '{print $2":"$6}'| cut -d: -f 2,4)
```

Procesos y Sistema operativo

Procesos en background

Se encargan de un servicio o tarea



Si se necesita reiniciar un servicio, se debe reiniciar el daemon, NO EL SISTEMA.

Generalmente se asocian a servicios del SO

pero pueden ser servicios de nivel de usuario

Cada sistema es distinto a otro, principalmente en base a sus daemons.

¹[https://en.wikipedia.org/wiki/Daemon_\(computing\)](https://en.wikipedia.org/wiki/Daemon_(computing))

Daemons

Ejemplos

cron

Daemon que ejecuta los crontab del sistema y de usuarios.

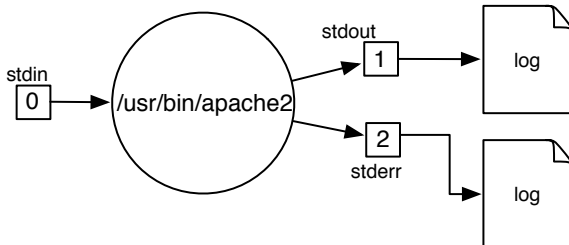
sshd

Daemon que ejecuta el servicio de acceso remoto SSH

apache2

Daemon que ejecuta el servidor web Apache

Debido a que operan en background, se considera una buena práctica que los daemon tengan su **stdout** y **stderr** redireccionados a un archivo, el que se denomina “log”

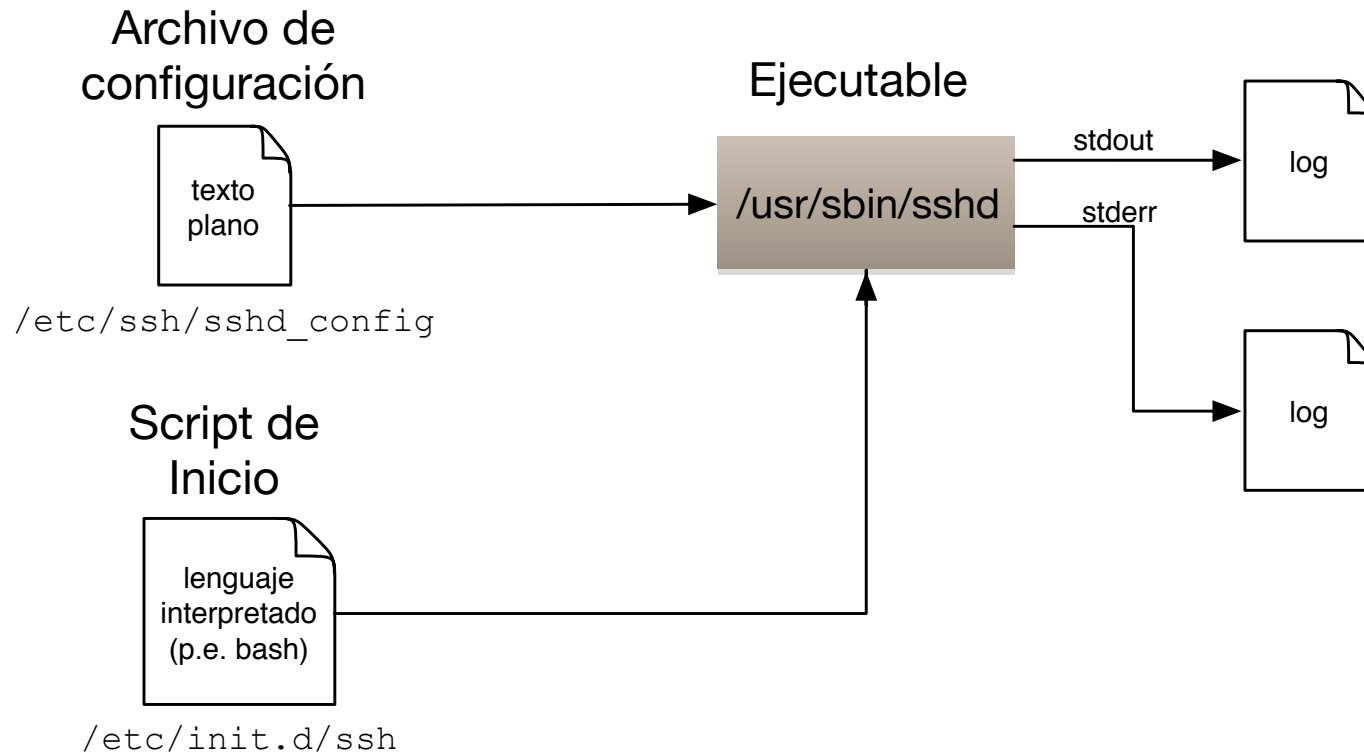


Revisar el log del daemon

Sirve para depurar
el comportamiento
del daemon

```
[root@server01:~# tail -10f /var/log/apache2/access.log
10.100.6.175 - - [07/Jan/2016:09:30:15 -0300] "GET /~gabriel/C/ejemplo.cgi HTTP/1.1" 200 202 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
10.100.6.175 - - [07/Jan/2016:16:55:44 -0300] "GET /~gabriel/C HTTP/1.1" 301 585 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
10.100.6.175 - - [07/Jan/2016:16:55:44 -0300] "GET /~gabriel/C/ HTTP/1.1" 200 796 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
10.100.6.175 - - [07/Jan/2016:16:55:44 -0300] "GET /icons/blank.gif HTTP/1.1" 200 431 "http://10.100.6.166/~gabriel/C/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
10.100.6.175 - - [07/Jan/2016:16:55:44 -0300] "GET /icons/text.gif HTTP/1.1" 200 513 "http://10.100.6.166/~gabriel/C/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
10.100.6.175 - - [07/Jan/2016:16:55:44 -0300] "GET /icons/unknown.gif HTTP/1.1" 200 529 "http://10.100.6.166/~gabriel/C/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
10.100.6.175 - - [07/Jan/2016:16:55:44 -0300] "GET /icons/back.gif HTTP/1.1" 200 500 "http://10.100.6.166/~gabriel/C/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
10.100.6.175 - - [07/Jan/2016:16:55:46 -0300] "GET /~gabriel/C/ HTTP/1.1" 200 797 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_2) AppleWebKit/601.3.9 (KHTML, like Gecko) Version/9.0.2 Safari/601.3.9"
```


Daemons: Arquitectura de operación



Old fashion

Ejecución directa del script de inicio

Ahora

Script de inicio llamado por **service** (8)

\$ `/etc/init.d/ssh start` \$ `service ssh start`

En Unix Sys V, el sistema tiene “perfiles” de daemons, llamados “runlevels”

Cada runlevel está representado por un directorio

/etc/rcX.d/

X representa en número del runlevel

Ejemplo de daemons que se inician en L2

```
gabriel@server01:~$ ls -l /etc/rc2.d/
total 4
-rw-r--r-- 1 root root 677 Apr  6  2015 README
lrwxrwxrwx 1 root root 18 Jul 22  2013 S01bootlogs -> ../init.d/bootlogs
lrwxrwxrwx 1 root root 14 Jul 22  2013 S01motd -> ../init.d/motd
lrwxrwxrwx 1 root root 17 Nov 12 16:39 S01rsyslog -> ../init.d/rsyslog
lrwxrwxrwx 1 root root 17 Dec 17 12:49 S02apache2 -> ../init.d/apache2
lrwxrwxrwx 1 root root 13 Dec 17 12:49 S03atd -> ../init.d/atd
lrwxrwxrwx 1 root root 14 Dec 17 12:49 S03cron -> ../init.d/cron
lrwxrwxrwx 1 root root 14 Dec 17 12:49 S03dbus -> ../init.d/dbus
lrwxrwxrwx 1 root root 17 Dec 17 12:49 S03postfix -> ../init.d/postfix
lrwxrwxrwx 1 root root 13 Dec 17 12:49 S03ssh -> ../init.d/ssh
lrwxrwxrwx 1 root root 18 Dec 17 12:49 S04rc.local -> ../init.d/rc.local
lrwxrwxrwx 1 root root 19 Dec 17 12:49 S04rmnologin -> ../init.d/rmnologin
```

Revisar contenido del archivo /etc/inittab

Automatización de tareas

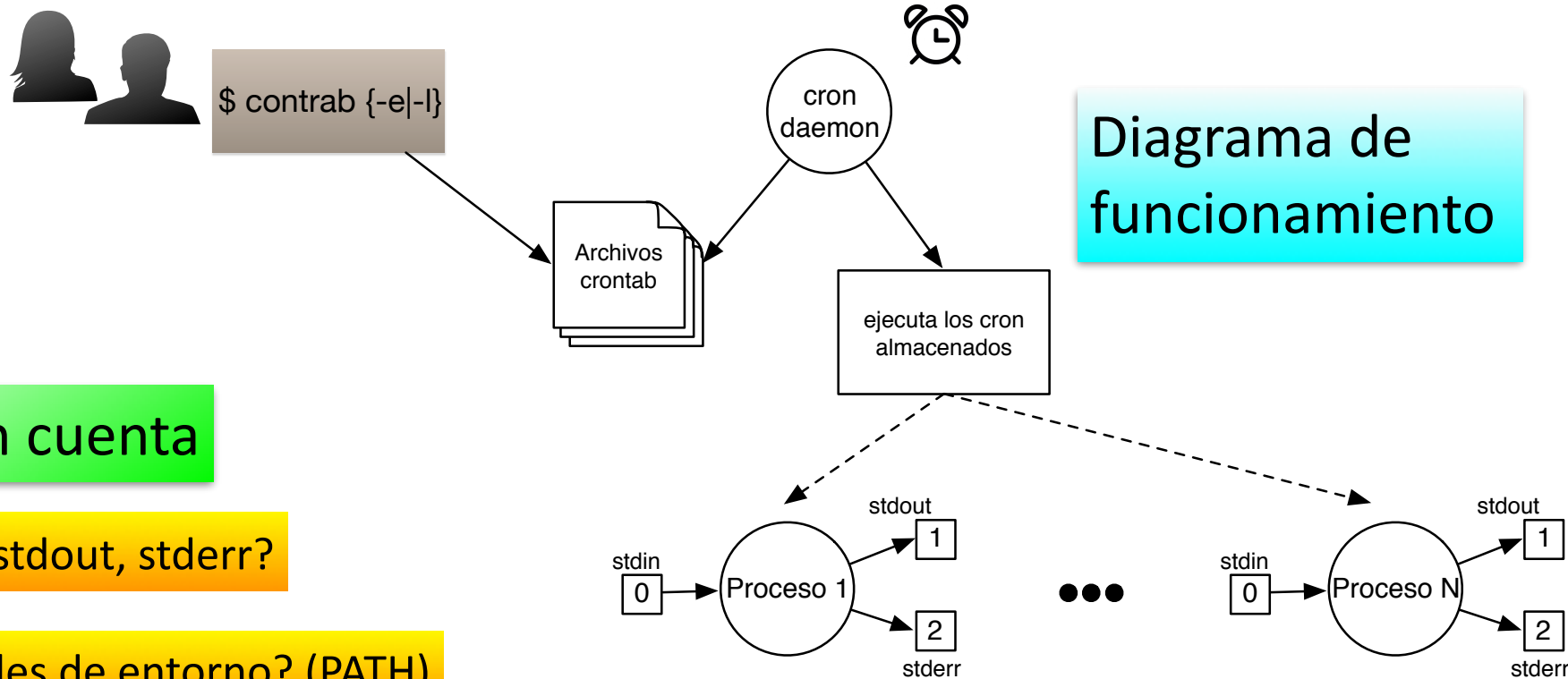
Crontab

Subsistema que permite calendarizar comandos.

Comando asociado: **crontab** (1)

Tiene una granularidad de 1 minuto.

Cada usuario puede tener varios crontabs



A tener en cuenta

¿stdin, stdout, stderr?

¿Variables de entorno? (PATH)

¿Debug?

Opciones del comando “crontab”

- e crea/modifica/elimina crontab. El editor que utiliza está definido en la variable `EDITOR`
- l lista los crontab del usuario

Sintaxis Básico de una entrada



Programación shell

Es un programa escrito en un lenguaje interpretado

Lenguajes de “Shell”: Bash, ash, ksh, sh, etc.

Otros: perl, php, python, ruby, etc.

Estructura

```
#!/bin/bash

#Comentario

variable=1
echo "hola mundo $1"
echo "variable=$variable"
```

Observaciones

No olvidar que el archivo debe tener permisos de ejecución.

Variables especiales

Variable	Significado
\$#	Cantidad de parámetros del script
\$n	Parámetro n-ésimo del script
\$?	Código de término del último comando
\$*	Lista de parámetros del script
\$_	PID del último proceso ejecutado
\$\$	PID del proceso actual


```
#!/bin/bash
SISTEMA="Ni idea";
FECHA=$(date +%Y-%m-%d,%H;%M);
cat << __FIN_AVISO__
==FECHA REPORTE: $FECHA ==
Sistema: $SISTEMA
=====
__FIN_AVISO__
```

