

Ejercicios resueltos de Despliegue de Aplicaciones Web

Los 31 ejercicios de Despliegue de Aplicaciones Web de la web, tema a tema: cada uno con su enunciado, los datos que necesitas (código de partida, ejemplos de entrada y salida o la base de datos) y, al final, la solución explicada.

Índice

Arquitecturas de Despliegue y la Nube

- [1. ¿Cuántos nuevos tiene tu arquitectura?](#)
- [3. Auditoría de un servidor recién instalado](#)

Servidores Web: Apache y Nginx

- [4. Generar un virtual host de Nginx](#)
- [6. Vigilar la caducidad de los certificados](#)
- [8. ¿Qué location atiende la petición?](#)

Servidores de Aplicaciones

- [9. Estado de los servicios](#)
- [11. Despliegue atómico con releases y vuelta atrás](#)

Contenedores con Docker

- [13. Revisar un Dockerfile](#)
- [15. Orden de arranque según depends on](#)

FTP, DNS y Servicios de Directorio

- [17. Leer una zona DNS](#)
- [19. Un resolvable DNS iterativo con caché](#)

Git, Documentación y CI/CD

- [21. La siguiente versión semántica](#)
- [23. Changelog a partir de los commits](#)
- [25. ¿Qué jobs ejecuta el pipeline?](#)

Ejercicios de Despliegue de Aplicaciones Web

- [26. Conflictos de puertos en Docker Compose](#)
- [28. Rotación de copias de seguridad](#)

Ejercicios largos

- [29. Informe de un registro de accesos web](#)
- [31. Despliegue por releases con enlace simbólico y vuelta atrás](#)

- [2. Un balanceador ponderado con caídas](#)

- [5. Estadísticas de un access.log](#)
- [7. Cadenas y bucles de redirecciones](#)

- [10. Revisar el server.xml de Tomcat](#)
- [12. Despliegue canary: ¿promocionar o revertir?](#)

- [14. Puertos de un docker compose](#)
- [16. ¿Qué capas se reconstruyen?](#)

- [18. Revisar una zona DNS antes de publicarla](#)
- [20. Evaluar filtros de búsqueda LDAP](#)

- [22. Una fusión a tres bandas](#)
- [24. Las métricas DORA de un equipo](#)

- [27. Informe de un pipeline de CI](#)

- [30. Analizador de la configuración de Nginx](#)

Arquitecturas de Despliegue y la Nube

1. ¿Cuántos nueves tiene tu arquitectura?

Medio · Arquitecturas de Despliegue y la Nube · apuntesdam.com/subject/despliegue/topic/arquitecturas-despliegue

La primera línea es «objetivo X» (la disponibilidad que exige el cliente, en %). Después, una línea por capa de la arquitectura: «capa nombre disponibilidad réplicas». Las réplicas de una capa están en paralelo (la capa solo cae si caen todas: $1 - (1 - a)^n$) y las capas están en serie (la disponibilidad total es el producto). Escribe por capa «nombre: $n \times a \% \rightarrow$ capa %» (a con 2 decimales, capa con 6), después «Total: ...» con 6 decimales, «Caída máxima al mes: X min» (mes de 43.200 minutos, 1 decimal) y «Cumple el objetivo» o «No cumple el objetivo del X %».

Código de partida (bash)

```
#!/bin/bash
export LC_ALL=C

# TODO: calcula con awk
cat > /dev/null
```

Ejemplo: Tienda con réplicas

ENTRADA

```
objetivo 99.95
capa balanceador 99.99 1
capa web 99.5 3
capa bd 99.9 2
```

SALIDA ESPERADA

```
balanceador: 1 x 99.99 % → 99.990000 %
web: 3 x 99.50 % → 99.999988 %
bd: 2 x 99.90 % → 99.999900 %
Total: 99.989888 %
Caída máxima al mes: 4.4 min
Cumple el objetivo
```

Ejemplo: Todo en un servidor

ENTRADA

```
objetivo 99.9
capa vps 99.9 1
capa bd 99.5 1
```

SALIDA ESPERADA

```
vps: 1 x 99.90 % → 99.900000 %
bd: 1 x 99.50 % → 99.500000 %
Total: 99.400500 %
Caída máxima al mes: 259.0 min
No cumple el objetivo del 99.90 %
```

2. Un balanceador ponderado con caídas

Difícil · Arquitecturas de Despliegue y la Nube · apuntesdam.com/subject/despliegue/topic/arquitecturas-despliegue

Nginx reparte las peticiones entre servidores con pesos distintos usando el algoritmo round robin ponderado «suave». Cada servidor tiene un peso w y un contador cw que empieza en 0. En cada petición: a todos los servidores en servicio se les suma su peso a cw ; se elige el de mayor cw (a igualdad, el primero de la lista) y a ese se le resta la suma de los pesos en servicio. La entrada tiene líneas «nombre peso», una línea «---» y después eventos: «peticion», «caido nombre» o «vuelve nombre» (al caer o volver, su cw se pone a 0). Escribe «→ nombre» por petición, «nombre fuera de servicio», «nombre de vuelta», «503 sin servidores» si no queda ninguno y, al final, «Reparto: $a=N$, $b=N\dots$ » en el orden de la lista.

Código de partida (bash)

```
#!/bin/bash
declare -A peso cw activo cuenta
orden=()
while read -r a b && [ "$a" != "---" ]; do
    orden+=("$a"); peso[$a]=$b; cw[$a]=0; activo[$a]=1; cuenta[$a]=0
done

while read -r evento nombre || [ -n "$evento" ]; do
    # TODO
    :
done

reparto=()
for s in "${orden[@]}"; do reparto+=("$s=${cuenta[$s]}"); done
(IFS=,; echo "Reparto: ${reparto[*]}" | sed 's/,/, /g')
```

Ejemplo: Pesos 5, 1 y 1

ENTRADA

```
a 5
b 1
c 1
---
peticion
peticion
peticion
peticion
peticion
peticion
peticion
```

SALIDA ESPERADA

```
→ a
→ a
→ b
→ a
→ c
→ a
→ a
Reparto: a=5, b=1, c=1
```

Ejemplo: Caídas y vueltas

ENTRADA

```
web1 2
web2 1
---
peticion
peticion
peticion
caido web1
peticion
peticion
caido web2
peticion
vuelve web1
peticion
peticion
```

SALIDA ESPERADA

```
→ web1
→ web2
→ web1
web1 fuera de servicio
→ web2
→ web2
web2 fuera de servicio
503 sin servidores
web1 de vuelta
→ web1
→ web1
Reparto: web1=4, web2=3
```

3. Auditoría de un servidor recién instalado

Muy difícil · Arquitecturas de Despliegue y la Nube · apuntesdam.com/subject/despliegue/topic/arquitecturas-despliegue

Antes de publicar una aplicación hay que bastionar el servidor. La entrada tiene tres bloques separados por «---»: el fichero `sshd_config`, la lista de puertos en escucha («protocolo dirección:puerto programa», como la resume `ss -tlnp`) y una línea «permitidos 22,80,443». Revisa, en este orden: `PermitRootLogin` yes (ALTO), `PasswordAuthentication` distinto de no (ALTO; si no aparece, su valor por defecto es yes), `MaxAuthTries` mayor que 4 (MEDIO; por defecto 6) y `X11Forwarding` yes (MEDIO). Ojo: en `sshd_config` las directivas no distinguen mayúsculas, se ignoran las líneas en blanco y los comentarios, y cuenta la PRIMERA aparición de cada directiva. Después, cada puerto escuchando en 0.0.0.0, `:::` o `*` que no esté permitido es un ALTO («puerto N (programa) abierto a internet»), por orden de número y sin repetir (los que escuchan en 127.0.0.1 o `:::1` no importan). Termina con «Resultado: N altos, M medios» o «Resultado: sin problemas».

Código de partida (bash)

```
#!/bin/bash
# TODO: analiza los tres bloques
cat > /dev/null
echo "Resultado: sin problemas"
```

Ejemplo: Instalación por defecto

ENTRADA

```
# Configuración de ejemplo
Port 22
PermitRootLogin yes
#PasswordAuthentication no
X11Forwarding yes
---
tcp 0.0.0.0:22 sshd
tcp 0.0.0.0:80 nginx
tcp 0.0.0.0:3306 mysqld
tcp [::]:3306 mysqld
tcp 127.0.0.1:6379 redis
---
permitidos 22,80,443
```

SALIDA ESPERADA

```
ALTO: PermitRootLogin yes permite entrar como
root con contraseña
ALTO: PasswordAuthentication permite contraseñas
(usa solo claves)
MEDIO: MaxAuthTries 6 (recomendado 4 o menos)
MEDIO: X11Forwarding activado sin necesidad
ALTO: puerto 3306 (mysqld) abierto a internet
Resultado: 3 altos, 2 medios
```

Ejemplo: Bien configurado

ENTRADA

```
PermitRootLogin no
PasswordAuthentication no
MaxAuthTries 3
---
tcp 0.0.0.0:22 sshd
tcp 0.0.0.0:443 nginx
tcp 127.0.0.1:3306 mysqld
tcp [::1]:5432 postgres
---
permitidos 22,80,443
```

SALIDA ESPERADA

```
Resultado: sin problemas
```

Servidores Web: Apache y Nginx

4. Generar un virtual host de Nginx

Fácil · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

La entrada es una línea «dominio raíz [puerto]». Genera el bloque server de Nginx con este formato exacto (4 espacios de sangría): «server {», «listen 80;», «server_name dominio www.dominio;» y, si no hay puerto, «root raíz;» y «index index.html;»; si hay puerto, en su lugar «location / {», «proxy_pass http://127.0.0.1:puerto;», «}». Termina con «}». Si falta el dominio o la raíz, escribe «Uso: dominio raíz [puerto]».

Código de partida (bash)

```
#!/bin/bash
read -r dominio raiz puerto

# TODO
exit 0
```

Ejemplo: Sitio estático

ENTRADA

```
apuntes.es /var/www/apuntes
```

SALIDA ESPERADA

```
server {
    listen 80;
    server_name apuntes.es www.apuntes.es;
    root /var/www/apuntes;
    index index.html;
}
```

Ejemplo: Aplicación con proxy

ENTRADA

```
tienda.com /srv/tienda 3000
```

SALIDA ESPERADA

```
server {
    listen 80;
    server_name tienda.com www.tienda.com;
    location / {
        proxy_pass http://127.0.0.1:3000;
    }
}
```

Ejemplo: Datos incompletos

ENTRADA

```
solo-dominio.com
```

SALIDA ESPERADA

```
Uso: dominio raíz [puerto]
```

5. Estadísticas de un access.log

Medio · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

La entrada es un fichero de registro de Nginx en formato combinado. Muestra «Peticiones: N», «Errores 4xx: N», «Errores 5xx: N» y «IP con más peticiones: IP (N)». Si hay empate, gana la IP menor alfabéticamente.

Código de partida (bash)

```
#!/bin/bash
log=$(cat)

# TODO
```

Ejemplo: Registro de ejemplo

ENTRADA

```
203.0.113.5 - - [30/Sep/2025:10:01:02 +0200] "GET / HTTP/1.1" 200 5120 "-" "Mozilla/5.0"
198.51.100.7 - - [30/Sep/2025:10:01:05 +0200] "GET /login HTTP/1.1" 200 2048 "-" "Mozilla/5.0"
203.0.113.5 - - [30/Sep/2025:10:01:09 +0200] "POST /login HTTP/1.1" 302 0 "-" "Mozilla/5.0"
192.0.2.44 - - [30/Sep/2025:10:02:11 +0200] "GET /wp-admin HTTP/1.1" 404 512 "-" "curl/8.0"
192.0.2.44 - - [30/Sep/2025:10:02:12 +0200] "GET /.env HTTP/1.1" 404 512 "-" "curl/8.0"
203.0.113.5 - - [30/Sep/2025:10:03:40 +0200] "GET /api/pedidos HTTP/1.1" 500 128 "-" "Mozilla/5.0"
198.51.100.7 - - [30/Sep/2025:10:04:01 +0200] "GET /panel HTTP/1.1" 403 256 "-" "Mozilla/5.0"
```

SALIDA ESPERADA

```
Peticiones: 7
Errores 4xx: 3
Errores 5xx: 1
IP con más peticiones: 203.0.113.5 (3)
```

6. Vigilar la caducidad de los certificados

Medio · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Certbot renueva solo, hasta que un día no lo hace (un cambio de DNS, un puerto cerrado) y la web muestra un aviso de seguridad a todos los visitantes. La primera línea es «hoy AAAA-MM-DD» y las siguientes, «dominio fecha-de-caducidad». Escribe los dominios ordenados por días restantes (a igualdad, alfabéticamente) con su estado: «CADUCADO hace N días» si ya ha caducado, «CRÍTICO: caduca en N días» si quedan 7 o menos, «AVISO: caduca en N días» si quedan 30 o menos y «ok (N días)» en otro caso. Al final, «Resumen: N críticos, M avisos» (los caducados cuentan como críticos).

Código de partida (bash)

```
#!/bin/bash
read -r _ hoy

# TODO
cat > /dev/null
```

Ejemplo: Varios dominios

ENTRADA

```
hoy 2026-09-30
apuntes.es 2026-12-01
tienda.com 2026-10-05
blog.tienda.com 2026-10-20
api.tienda.com 2026-09-25
```

SALIDA ESPERADA

```
api.tienda.com: CADUCADO hace 5 días
tienda.com: CRÍTICO: caduca en 5 días
blog.tienda.com: AVISO: caduca en 20 días
apuntes.es: ok (62 días)
Resumen: 2 críticos, 1 avisos
```

Ejemplo: Empates y límites

ENTRADA

```
hoy 2026-02-20
b.es 2026-03-22
a.es 2026-03-22
c.es 2026-02-27
d.es 2026-02-20
e.es 2026-03-23
```

SALIDA ESPERADA

```
d.es: CRÍTICO: caduca en 0 días
c.es: CRÍTICO: caduca en 7 días
a.es: AVISO: caduca en 30 días
b.es: AVISO: caduca en 30 días
e.es: ok (31 días)
Resumen: 2 críticos, 2 avisos
```

7. Cadenas y bucles de redirecciones

Difícil · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Tras varias migraciones de una web, las redirecciones 301 se acumulan: /antiguo → /intermedio → /nuevo obliga al navegador (y a Google) a dar varios saltos, y un error puede crear un bucle infinito. La entrada tiene reglas «origen destino» (una por línea), una línea «---» y después URLs a comprobar. Para cada URL sigue las reglas y escribe: «url: 200 sin redirección» si no hay regla, «url: 301 → destino» si hay un solo salto, «url: CADENA de N saltos → final (redirige directamente)» si hay más, o «url: BUCLE a → b → ... → a» si vuelve a una URL ya visitada (mostrando el camino hasta la repetición).

Código de partida (bash)

```
#!/bin/bash
declare -A regla
while read -r origen destino && [ "$origen" != "---" ]; do
    regla[$origen]=$destino
done

while read -r url || [ -n "$url" ]; do
    [ -z "$url" ] && continue
    # TODO
    echo "$url: 200 sin redirección"
done
```

Ejemplo: Migraciones acumuladas

ENTRADA

```
/blog /noticias
/noticias /actualidad
/actualidad /revista
/tienda /productos
---
/blog
/tienda
/contacto
/noticias
```

SALIDA ESPERADA

```
/blog: CADENA de 3 saltos → /revista (redirige
directamente)
/tienda: 301 → /productos
/contacto: 200 sin redirección
/noticias: CADENA de 2 saltos → /revista
(redirige directamente)
```

Ejemplo: Un bucle

ENTRADA

```
/a /b
/b /c
/c /a
/x /x
/inicio /a
---
/inicio
/x
/b
```

SALIDA ESPERADA

```
/inicio: BUCLE /inicio → /a → /b → /c → /a
/x: BUCLE /x → /x
/b: BUCLE /b → /c → /a → /b
```

8. ¿Qué location atiende la petición?

Muy difícil · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Entender cómo elige Nginx el bloque location evita horas de depuración. La entrada tiene bloques location («= /ruta», «^~ /prefijo», «~ expresión», «~* expresión» o solo «/prefijo»), una línea «---» y URIs. Aplica el algoritmo de Nginx: 1) si una location exacta (=) coincide, gana; 2) si no, busca el prefijo (normal o ^~) más largo que coincida; si ese prefijo es ^~, gana sin mirar las expresiones; 3) si no, prueba las expresiones regulares (~ distingue mayúsculas, ~* no) en el orden en que aparecen y gana la primera que coincida; 4) si ninguna coincide, gana el prefijo más largo encontrado en el paso 2. Escribe «uri → location» tal como está escrita (el prefijo normal sin modificador), o «uri → 404 (ninguna)».

Código de partida (bash)

```
#!/bin/bash
mods=(); pats=()
while read -r a b && [ "$a" != "---" ]; do
    if [ -n "$b" ]; then mods+=("$a"); pats+=("$b"); else mods+=(""); pats+=("$a"); fi
done

while read -r uri || [ -n "$uri" ]; do
    [ -z "$uri" ] && continue
    # TODO: aplica el algoritmo de Nginx
    echo "$uri → 404 (ninguna)"
done
```

Ejemplo: Configuración típica

ENTRADA

```
= /  
/  
/api/  
^~ /static/  
~ \.php$  
~* \.(jpg|png)$  
---  
/  
/contacto  
/api/usuarios  
/static/logo.png  
/imagenes/FOTO.JPG  
/admin/index.php  
/api/export.php
```

SALIDA ESPERADA

```
/ → = /  
/contacto → /  
/api/usuarios → /api/  
/static/logo.png → ^~ /static/  
/imagenes/FOTO.JPG → ~* \.(jpg|png)$  
/admin/index.php → ~ \.php$  
/api/export.php → ~ \.php$
```

Ejemplo: Prefijo más largo y regex

ENTRADA

```
/docs/  
/docs/v2/  
~ ^/docs/v[0-9]+/privado  
---  
/docs/intro  
/docs/v2/guia  
/docs/v2/privado/claves  
/blog
```

SALIDA ESPERADA

```
/docs/intro → /docs/  
/docs/v2/guia → /docs/v2/  
/docs/v2/privado/claves → ~ ^/docs/v[0-9]+/privado  
/blog → 404 (ninguna)
```

Servidores de Aplicaciones

9. Estado de los servicios

Medio · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

Cada línea es el resultado de comprobar un servicio: «nombre código_http milisegundos». Clasifica cada uno como «OK» (200 y menos de 500 ms), «LENTO» (200 y 500 ms o más) o «CAÍDO» (cualquier otro código), con el formato «nombre: ESTADO». Al final, «Estado general: OK», «DEGRADADO» (si hay alguno lento y ninguno caído) o «CAÍDO» (si hay alguno caído).

Código de partida (bash)

```
#!/bin/bash  
while read -r nombre codigo ms || [ -n "$nombre" ]; do  
    # TODO  
    :  
done
```

Ejemplo: Todo bien

ENTRADA

```
web 200 120  
api 200 340
```

SALIDA ESPERADA

```
web: OK  
api: OK  
Estado general: OK
```

Ejemplo: Uno lento

ENTRADA

```
web 200 120
api 200 850
cola 200 499
```

SALIDA ESPERADA

```
web: OK
api: LENTO
cola: OK
Estado general: DEGRADADO
```

Ejemplo: Uno caído

ENTRADA

```
web 200 120
api 503 30
pagos 200 900
```

SALIDA ESPERADA

```
web: OK
api: CAÍDO
pagos: LENTO
Estado general: CAÍDO
```

10. Revisar el server.xml de Tomcat

Medio · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

La entrada es un fichero server.xml (cada elemento en una línea). Muestra, en el orden en que aparecen: «Apagado: puerto P, orden O» (elemento Server), «Conector P PROTOCOLO» con « (TLS)» si tiene SSLEnabled="true" (si no hay protocolo, es HTTP/1.1) y «Host nombre → appBase (autoDeploy=valor)» (si no se indica, autoDeploy vale true). Ignora las líneas de comentario (<!-- ... -->). Después escribe los avisos, en el orden en que se detectan: «AVISO: puerto de apagado P con la orden por defecto» si el puerto no es -1 y la orden es SHUTDOWN; «PELIGRO: conector AJP P accesible desde fuera (Ghostcat)» si un conector AJP no tiene address local (127.0.0.1, ::1 o localhost); «AVISO: autoDeploy activado en nombre» para cada host que lo tenga. Termina con «Avisos: N».

Código de partida (bash)

```
#!/bin/bash
# TODO: inventario y avisos
cat > /dev/null
echo "Avisos: 0"
```

Ejemplo: Instalación por defecto

ENTRADA

```
<?xml version="1.0" encoding="UTF-8"?>
<Server port="8005" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8080" protocol="HTTP/1.1"
      connectionTimeout="20000" redirectPort="8443" />
    <!-- <Connector port="8443"
      protocol="HTTP/1.1" SSLEnabled="true" /> -->
    <Connector port="8009" protocol="AJP/1.3"
      redirectPort="8443" />
    <Engine name="Catalina"
      defaultHost="localhost">
      <Host name="localhost" appBase="webapps"
        unpackWARs="true" autoDeploy="true">
      </Host>
    </Engine>
  </Service>
</Server>
```

SALIDA ESPERADA

```
Apagado: puerto 8005, orden SHUTDOWN
Conector 8080 HTTP/1.1
Conector 8009 AJP/1.3
Host localhost → webapps (autoDeploy=true)
AVISO: puerto de apagado 8005 con la orden por defecto
PELIGRO: conector AJP 8009 accesible desde fuera (Ghostcat)
AVISO: autoDeploy activado en localhost
Avisos: 3
```

Ejemplo: Bien configurado

ENTRADA

```
<Server port="-1" shutdown="SHUTDOWN">
  <Service name="Catalina">
    <Connector port="8443"
protocol="org.apache.coyote.http11.Http11NioProtocol"
SSLEnabled="true" maxThreads="200" />
    <Connector port="8009" protocol="AJP/1.3"
address="127.0.0.1" secret="s3cr3t" />
    <Engine name="Catalina"
defaultHost="tienda.es">
      <Host name="tienda.es"
appBase="/srv/tienda" autoDeploy="false" />
      <Host name="intranet.tienda.es"
appBase="/srv/intranet" autoDeploy="false" />
    </Engine>
  </Service>
</Server>
```

SALIDA ESPERADA

```
Apagado: puerto -1, orden SHUTDOWN
Conector 8443
org.apache.coyote.http11.Http11NioProtocol (TLS)
Conector 8009 AJP/1.3
Host tienda.es → /srv/tienda (autoDeploy=false)
Host intranet.tienda.es → /srv/intranet
(autoDeploy=false)
Avisos: 0
```

11. Despliegue atómico con releases y vuelta atrás

Difícil · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

Cada versión se copia en su propia carpeta releases/VERSIÓN y la web apunta a un enlace simbólico current: cambiarlo es instantáneo y volver atrás es apuntarlo a la anterior. Usa carpetas y enlaces reales dentro de /tmp/app. Cada línea es una orden: «desplegar v» (crea releases/v, apunta current a ella y conserva solo las 3 releases más recientes, sin borrar nunca la actual; si v ya existe, «Error: v ya existe»), «rollback» (apunta current a la release desplegada justo antes de la actual; si no hay, «Error: no hay versión anterior») y «estado». Tras cada cambio y con «estado», escribe «current → v (releases: a b c)» con las releases de más antigua a más reciente (o «Sin desplegar»).

Código de partida (bash)

```
#!/bin/bash
base=/tmp/app
rm -rf "$base" && mkdir -p "$base/releases"
orden=()

while read -r accion version || [ -n "$accion" ]; do
  # TODO
  echo "Sin desplegar"
done
```

Ejemplo: Varias versiones

ENTRADA

```
estado
desplegar v1
desplegar v2
desplegar v3
desplegar v4
rollback
rollback
estado
```

SALIDA ESPERADA

```
Sin desplegar
current → v1 (releases: v1)
current → v2 (releases: v1 v2)
current → v3 (releases: v1 v2 v3)
current → v4 (releases: v2 v3 v4)
current → v3 (releases: v2 v3 v4)
current → v2 (releases: v2 v3 v4)
current → v2 (releases: v2 v3 v4)
```

Ejemplo: Errores

ENTRADA

```
rollback
desplegar a
rollback
desplegar b
desplegar b
rollback
desplegar c
desplegar d
desplegar e
```

SALIDA ESPERADA

```
Error: no hay versión anterior
current → a (releases: a)
Error: no hay versión anterior
current → b (releases: a b)
Error: b ya existe
current → a (releases: a b)
current → c (releases: a b c)
current → d (releases: b c d)
current → e (releases: c d e)
```

12. Despliegue canary: ¿promocionar o revertir?

Muy difícil · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

En un despliegue canary, la versión nueva recibe primero un 5 % del tráfico, luego un 25 %, un 50 % y por fin el 100 %, y solo avanza si se comporta al menos tan bien como la estable. Cada línea es una evaluación con las métricas de ambas versiones: «peticiones errores p95 | peticiones errores p95» (estable | canario; p95 en milisegundos). En cada evaluación, en este orden: si el canario tiene menos de 50 peticiones, «N: X % → datos insuficientes (P peticiones)» y se sigue igual; si su tasa de errores (en %) supera en más de 0,5 puntos a la estable, «N: X % → REVERTIDA (errores A % frente a B %)» con 2 decimales, y se termina; si su p95 supera en más de un 20 % al estable, «N: X % → REVERTIDA (latencia A ms frente a B ms)» y se termina; si todo va bien, «N: X % → ok, pasa al Y %» o, si ya estaba al 100 %, «N: 100 % → ok, PROMOCIONADA» y se termina. Si se acaban las evaluaciones sin decidir, «Sin decisión: se mantiene el X %».

Código de partida (bash)

```
#!/bin/bash
export LC_ALL=C
# TODO: evalúa cada línea con awk
cat > /dev/null
```

Ejemplo: Todo va bien

ENTRADA

```
10000 20 180 | 520 1 190
8000 16 175 | 2600 6 200
6000 12 170 | 6000 15 180
2000 5 160 | 10000 30 185
```

SALIDA ESPERADA

```
1: 5 % → ok, pasa al 25 %
2: 25 % → ok, pasa al 50 %
3: 50 % → ok, pasa al 100 %
4: 100 % → ok, PROMOCIONADA
```

Ejemplo: Errores y datos insuficientes

ENTRADA

```
9000 45 200 | 30 0 150
9000 45 200 | 480 3 210
7000 30 190 | 2400 25 195
```

SALIDA ESPERADA

```
1: 5 % → datos insuficientes (30 peticiones)
2: 5 % → ok, pasa al 25 %
3: 25 % → REVERTIDA (errores 1.04 % frente a 0.43 %)
```

Contenedores con Docker

13. Revisar un Dockerfile

Medio · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

La entrada es un Dockerfile. Escribe un aviso por línea, en este orden, cuando corresponda: «Imagen base sin versión concreta» (algún FROM sin etiqueta o con :latest), «apt-get install sin limpiar la caché» (un RUN con apt-get install sin rm -rf /var/lib/apt/lists), «Se copia todo el código antes de instalar las dependencias» (un COPY . . aparece antes del RUN que ejecuta npm install/npm ci, composer install o pip install) y «El contenedor se ejecuta como root» (no hay ninguna instrucción USER). Si no hay ningún aviso, escribe «Sin avisos».

Código de partida (bash)

```
#!/bin/bash
dockerfile=$(cat)
avisos=()

# TODO
```

Ejemplo: Dockerfile mejorable

ENTRADA

```
FROM node
WORKDIR /app
COPY . .
RUN npm install
CMD ["node", "server.js"]
```

SALIDA ESPERADA

```
Imagen base sin versión concreta
Se copia todo el código antes de instalar las
dependencias
El contenedor se ejecuta como root
```

Ejemplo: Dockerfile correcto

ENTRADA

```
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
USER node
CMD ["node", "server.js"]
```

SALIDA ESPERADA

```
Sin avisos
```

Ejemplo: apt-get y latest

ENTRADA

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y nginx
USER www-data
```

SALIDA ESPERADA

```
Imagen base sin versión concreta
apt-get install sin limpiar la caché
```

14. Puertos de un docker compose

Medio · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

Cada línea describe los puertos de un servicio: «servicio: p1 p2 ...», donde cada puerto es «anfitrión:contenedor» (se publica) o solo «contenedor» (queda en la red interna). Para cada puerto, en orden: si es solo interno, «servicio: P solo interno (no se publica)»; si alguno de los números no es un entero entre 1 y 65535, «servicio: P ERROR puerto no válido»; si el puerto del anfitrión ya lo publica otro servicio, «servicio: P ERROR el puerto H ya lo publica otro»; si no, «servicio: H → C», con « (privilegiado)» si el del anfitrión es menor que 1024. Al final, «Publicados: N, errores: M».

Código de partida (bash)

```
#!/bin/bash
declare -A usado
errores=0
while IFS= read -r linea || [ -n "$linea" ]; do
    [ -z "$linea" ] && continue
    servicio=${linea%:*}
    resto=${linea#*:}
    for p in $resto; do
        # TODO
        echo "$servicio: $p"
    done
done
echo "Publicados: ${#usado[@]}, errores: $errores"
```

Ejemplo: Proxy, aplicación y base de datos

ENTRADA

```
proxy: 80:80 443:443
app: 3000
db: 3306
adminer: 8080:8080
```

SALIDA ESPERADA

```
proxy: 80 → 80 (privilegiado)
proxy: 443 → 443 (privilegiado)
app: 3000 solo interno (no se publica)
db: 3306 solo interno (no se publica)
adminer: 8080 → 8080
Publicados: 3, errores: 0
```

Ejemplo: Conflictos y errores

ENTRADA

```
web: 8080:80
api: 8080:3000 8081:3000
cola: 70000:5672 abc:15672
metricas: 9090:9090 443:9443
```

SALIDA ESPERADA

```
web: 8080 → 80
api: 8080:3000 ERROR el puerto 8080 ya lo publica web
api: 8081 → 3000
cola: 70000:5672 ERROR puerto no válido
cola: abc:15672 ERROR puerto no válido
metricas: 9090 → 9090
metricas: 443 → 9443 (privilegiado)
Publicados: 4, errores: 3
```

15. Orden de arranque según depends_on

Difícil · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

Docker Compose arranca primero los servicios de los que dependen otros. Cada línea es «servicio: dep1 dep2 ...» (puede no tener dependencias). Calcula el arranque por rondas: en cada ronda arrancan a la vez, en orden alfabético, todos los servicios cuyas dependencias ya han arrancado («Ronda N: a b c»). Antes, comprueba dos errores: si un servicio depende de uno que no existe, escribe solo «Error: s depende de d, que no existe» (el primero que encuentres, en el orden de las líneas y de sus dependencias); si hay dependencias circulares, escribe solo «Ciclo: a → b → ... → a», buscando con una búsqueda en profundidad que visita los servicios y sus dependencias en orden alfabético y muestra el ciclo desde el primer servicio repetido.

Código de partida (bash)

```
#!/bin/bash
declare -A dep
orden=()
while IFS=: read -r s d || [ -n "$s" ]; do
    [ -z "$s" ] && continue
    s=$(echo $s); dep[$s]=$d; orden+=("$s")
done

# TODO: errores, ciclos y rondas
echo "Ronda 1: ${orden[*]}"
```

Ejemplo: Tienda completa

ENTRADA

```
proxy: app
app: db cache
cache:
db:
worker: db cache
migraciones: db
```

SALIDA ESPERADA

```
Ronda 1: cache db
Ronda 2: app migraciones worker
Ronda 3: proxy
```

Ejemplo: Un ciclo

ENTRADA

```
api: auth
auth: usuarios
usuarios: api
web: api
```

SALIDA ESPERADA

```
Ciclo: api → auth → usuarios → api
```

Ejemplo: Dependencia inexistente

ENTRADA

```
app: db redis
db:
```

SALIDA ESPERADA

```
Error: app depende de redis, que no existe
```

16. ¿Qué capas se reconstruyen?

Muy difícil · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

Docker reutiliza cada capa de la construcción anterior si la instrucción es idéntica, si en las instrucciones COPY o ADD no ha cambiado ninguno de los ficheros que copian y si ninguna capa anterior se ha reconstruido (en cuanto una se reconstruye, todas las siguientes también). La entrada tiene el Dockerfile de la construcción anterior, «---», el nuevo, «---» y la lista de ficheros modificados. Una fuente de COPY puede ser «.» (todo), un fichero, una carpeta (acabada o no en /: cubre lo que hay dentro) o un patrón con * (package*.json). El último argumento de COPY es el destino, y un COPY --from=... copia de otra etapa: no depende de los ficheros. Escribe por cada instrucción del nuevo Dockerfile «N instrucción → caché» o «N instrucción → RECONSTRUIR (motivo)», con motivo «instrucción cambiada», «cambió fichero» (el primer fichero modificado de la lista que la afecte) o «capa anterior reconstruida», y al final «Capas reutilizadas: X de N».

Código de partida (bash)

```
#!/bin/bash
mapfile -t todo
bloque=0; v1=(); v2=(); cambiados=()
for l in "${todo[@]"; do
    if [ "$l" = "---" ]; then ((bloque++)); continue; fi
    [ -z "$l" ] && continue
    case $bloque in 0) v1+=("$l" );; 1) v2+=("$l" );; 2) cambiados+=("$l" );; esac
done

# TODO: decide caché o reconstrucción para cada instrucción de v2
for i in "${!v2[@]"; do echo "$((i + 1)) ${v2[i]} → caché"; done
echo "Capas reutilizadas: ${#v2[@]} de ${#v2[@]}"
```

Ejemplo: Buen orden: solo cambia el código

ENTRADA

```
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
---
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
---
src/app.js
```

SALIDA ESPERADA

```
1 FROM node:22-alpine → caché
2 WORKDIR /app → caché
3 COPY package*.json ./ → caché
4 RUN npm ci → caché
5 COPY . . → RECONSTRUIR (cambió src/app.js)
6 RUN npm run build → RECONSTRUIR (capa anterior
reconstruida)
Capas reutilizadas: 4 de 6
```

Ejemplo: Mal orden: se reinstalan las dependencias

ENTRADA

```
FROM node:22-alpine
WORKDIR /app
COPY . .
RUN npm ci
RUN npm run build
---
FROM node:22-alpine
WORKDIR /app
COPY . .
RUN npm ci
RUN npm run build
---
src/app.js
```

SALIDA ESPERADA

```
1 FROM node:22-alpine → caché
2 WORKDIR /app → caché
3 COPY . . → RECONSTRUIR (cambió src/app.js)
4 RUN npm ci → RECONSTRUIR (capa anterior
reconstruida)
5 RUN npm run build → RECONSTRUIR (capa anterior
reconstruida)
Capas reutilizadas: 2 de 5
```

Ejemplo: Cambian las dependencias y una instrucción

ENTRADA

```
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build
---
FROM node:22-alpine
WORKDIR /app
COPY package*.json ./
RUN npm ci --omit=dev
COPY . .
RUN npm run build
---
package-lock.json
README.md
```

SALIDA ESPERADA

```
1 FROM node:22-alpine → caché
2 WORKDIR /app → caché
3 COPY package*.json ./ → RECONSTRUIR (cambió
package-lock.json)
4 RUN npm ci --omit=dev → RECONSTRUIR (capa
anterior reconstruida)
5 COPY . . → RECONSTRUIR (capa anterior
reconstruida)
6 RUN npm run build → RECONSTRUIR (capa anterior
reconstruida)
Capas reutilizadas: 2 de 6
```

17. Leer una zona DNS

Medio · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

La entrada es un fichero de zona de BIND con una directiva \$ORIGIN. Muestra los registros A, AAAA, CNAME y MX, en su orden, con el nombre completo: «www.tienda.com. CNAME tienda.com.». «@» es el propio origen; un nombre que no termina en punto es relativo y se le añade el origen (también en el destino de un CNAME). En los MX muestra la prioridad y el servidor: «tienda.com. MX 10 correo.tienda.com.». Ignora comentarios (;), directivas y otros tipos. Termina con «Registros: N».

Código de partida (bash)

```
#!/bin/bash
# TODO: procesa la zona con awk
cat > /dev/null
```

Ejemplo: Zona de la tienda

ENTRADA

```
$ORIGIN tienda.com.
$TTL 3600
@      IN  SOA ns1.tienda.com. admin.tienda.com.
(2025093001 7200 3600 1209600 3600)
@      IN  NS   ns1.tienda.com.
@      IN  A    203.0.113.10
www    IN  CNAME @
api    IN  A    203.0.113.20
api    IN  AAAA 2001:db8::20
@      IN  MX 10 correo.tienda.com.
correo IN  A    203.0.113.30
; comentario: servidor antiguo
ftp.otro.org. IN A 198.51.100.1
```

SALIDA ESPERADA

```
tienda.com. A 203.0.113.10
www.tienda.com. CNAME tienda.com.
api.tienda.com. A 203.0.113.20
api.tienda.com. AAAA 2001:db8::20
tienda.com. MX 10 correo.tienda.com.
correo.tienda.com. A 203.0.113.30
ftp.otro.org. A 198.51.100.1
Registros: 7
```

18. Revisar una zona DNS antes de publicarla

Medio · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

Un error en la zona puede dejar sin web o sin correo a toda una empresa. La entrada es un fichero de zona con \$ORIGIN (ignora comentarios, líneas vacías y otras directivas; cada registro es «nombre IN tipo datos...»; @ es el origen y los nombres sin punto final son relativos). Comprueba, escribiendo los mensajes en este orden: 1) para cada nombre, por orden de aparición, «ERROR: CNAME en el ápice (nombre)» si el propio dominio tiene un CNAME, o «ERROR: nombre tiene un CNAME y otros registros» si un nombre con CNAME tiene además otros registros; 2) para cada MX, «ERROR: el MX de nombre apunta a destino, que es un CNAME» (el destino de un MX debe tener registros A o AAAA); 3) para cada CNAME, MX o NS cuyo destino no acabe en punto pero termine en el dominio (o sea el dominio), «AVISO: falta el punto final en «destino» (se leería destino.origen)». Termina con «Errores: N, avisos: M».

Código de partida (bash)

```
#!/bin/bash
# TODO: comprueba la zona con awk
cat > /dev/null
echo "Errores: 0, avisos: 0"
```

Ejemplo: Zona con fallos

ENTRADA

```
$ORIGIN tienda.es.  
$TTL 3600  
@ IN SOA ns1.tienda.es. admin.tienda.es.  
(2026093001 7200 3600 1209600 3600)  
@ IN NS ns1.tienda.es.  
@ IN A 203.0.113.10  
www IN CNAME @  
correo IN CNAME www  
@ IN MX 10 correo  
blog IN CNAME tienda.es  
blog IN TXT "verificacion"  
ns1 IN A 203.0.113.2
```

SALIDA ESPERADA

```
ERROR: blog.tienda.es. tiene un CNAME y otros  
registros  
ERROR: el MX de tienda.es. apunta a  
correo.tienda.es., que es un CNAME  
AVISO: falta el punto final en «tienda.es» (se  
leería tienda.es.tienda.es.)  
Errores: 2, avisos: 1
```

Ejemplo: Zona correcta

ENTRADA

```
$ORIGIN ejemplo.org.  
; zona correcta  
@ IN NS ns1.proveedor.net.  
@ IN A 192.0.2.1  
www IN CNAME @  
@ IN MX 10 mail  
mail IN A 192.0.2.25
```

SALIDA ESPERADA

```
Errores: 0, avisos: 0
```

19. Un resolvidor DNS iterativo con caché

Difícil · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

Así trabaja el DNS de tu operadora. La entrada tiene líneas «servidor nombre tipo valor ttl» con lo que sabe cada servidor (el servidor raíz se llama raíz; los demás, por su nombre), una línea «---» y consultas «segundo nombre». Para resolver un nombre: si su registro A está en caché y no ha caducado (caduca en segundo + ttl), responde «(caché)» sin preguntar a nadie. Si no, empieza por el servidor de la delegación (NS) en caché y vigente más específica cuyo dominio contenga el nombre (o por raíz si no hay ninguna) y pregunta: si ese servidor tiene el A del nombre, lo guarda en caché y responde; si tiene un NS de un dominio que contiene el nombre, guarda esa delegación en caché y pasa a preguntar al servidor indicado (si hay varios, el del dominio más largo); si no, el nombre no existe. Escribe «segundo nombre: IP (N consultas)», «... (caché)» o «... no existe (N consultas)» y, al final, «Consultas totales: N».

Código de partida (bash)

```
#!/bin/bash  
declare -A registro          # "servidor|nombre|tipo" -> "valor ttl"  
declare -A delegaciones      # servidor -> lista de "dominio|servidor_hijo|ttl"  
while read -r srv nom tipo val ttl && [ "$srv" != "---" ]; do  
    if [ "$tipo" = "NS" ]; then delegaciones[$srv]+="$nom|$val|$ttl "; else  
        registro["$srv|$nom|$tipo"]="$val $ttl"; fi  
done  
  
total=0  
while read -r t nom || [ -n "$t" ]; do  
    [ -z "$t" ] && continue  
    # TODO  
    echo "$t $nom: no existe (0 consultas)"  
done  
echo "Consultas totales: $total"
```

Ejemplo: Caché de respuestas y delegaciones

ENTRADA

```
raiz es. NS a.nic.es. 172800
raiz org. NS a0.org. 172800
a.nic.es. tienda.es. NS ns1.tienda.es. 3600
a.nic.es. academia.es. NS ns.academia.es. 3600
ns1.tienda.es. www.tienda.es. A 203.0.113.10 300
ns1.tienda.es. api.tienda.es. A 203.0.113.20 60
ns.academia.es. academia.es. A 198.51.100.7 600
---
0 www.tienda.es.
10 api.tienda.es.
20 www.tienda.es.
100 api.tienda.es.
400 www.tienda.es.
```

SALIDA ESPERADA

```
0 www.tienda.es.: 203.0.113.10 (3 consultas)
10 api.tienda.es.: 203.0.113.20 (1 consultas)
20 www.tienda.es.: 203.0.113.10 (caché)
100 api.tienda.es.: 203.0.113.20 (1 consultas)
400 www.tienda.es.: 203.0.113.10 (1 consultas)
Consultas totales: 6
```

Ejemplo: Otros dominios y nombres inexistentes

ENTRADA

```
raiz es. NS a.nic.es. 172800
raiz org. NS a0.org. 172800
a.nic.es. tienda.es. NS ns1.tienda.es. 3600
a.nic.es. academia.es. NS ns.academia.es. 3600
ns1.tienda.es. www.tienda.es. A 203.0.113.10 300
ns1.tienda.es. api.tienda.es. A 203.0.113.20 60
ns.academia.es. academia.es. A 198.51.100.7 600
---
500 academia.es.
700 ftp.tienda.es.
800 www.ejemplo.org.
900 academia.es.
```

SALIDA ESPERADA

```
500 academia.es.: 198.51.100.7 (3 consultas)
700 ftp.tienda.es.: no existe (2 consultas)
800 www.ejemplo.org.: no existe (2 consultas)
900 academia.es.: 198.51.100.7 (caché)
Consultas totales: 7
```

20. Evaluar filtros de búsqueda LDAP

Muy difícil · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

Las aplicaciones buscan en el directorio de la empresa con filtros LDAP en notación prefija: (uid=ana), (&(objectClass=person)(ou=ventas)), ((ou=ventas)(ou=soporte)), (!(uid=admin)), (mail=*) (el atributo existe) y comodines como (cn=Ana*) o (mail=*@tienda.es). La entrada tiene entradas del directorio («atributo: valor» por línea, separadas por líneas en blanco; un atributo puede repetirse, como objectClass o memberOf), una línea «---» y filtros, uno por línea. Para cada filtro escribe «filtro → uid1, uid2» con los uid de las entradas que lo cumplen, en el orden del directorio, o «filtro → (ninguna)». Los nombres de atributo y las comparaciones de valores no distinguen mayúsculas; un atributo con varios valores cumple la condición si la cumple alguno.

Código de partida (bash)

```
#!/bin/bash
# TODO: lee las entradas y evalúa cada filtro con awk
cat > /dev/null
```

Ejemplo: Filtros básicos

ENTRADA

```
dn: uid=ana,ou=personas,dc=tienda,dc=es
objectClass: person
objectClass: inetOrgPerson
uid: ana
cn: Ana García
ou: ventas
mail: ana@tienda.es
memberOf: cn=empleados,ou=grupos,dc=tienda,dc=es

dn: uid=luis,ou=personas,dc=tienda,dc=es
objectClass: person
uid: luis
cn: Luis Pérez
ou: soporte
mail: luis@gmail.com
memberOf: cn=empleados,ou=grupos,dc=tienda,dc=es
memberOf: cn=admins,ou=grupos,dc=tienda,dc=es

dn: uid=admin,dc=tienda,dc=es
objectClass: account
uid: admin

dn: uid=eva,ou=personas,dc=tienda,dc=es
objectClass: person
uid: eva
cn: Eva Ruiz
ou: Ventas
mail: eva@tienda.es
---
(uid=ana)
(ou=ventas)
(&(objectClass=person)(mail=*))
(!&(objectClass=person))
```

SALIDA ESPERADA

```
(uid=ana) → ana
(ou=ventas) → ana, eva
(&(objectClass=person)(mail=*)) → ana, luis, eva
(!&(objectClass=person)) → admin
```

Ejemplo: Comodines y combinaciones

ENTRADA

```
dn: uid=ana,ou=personas,dc=tienda,dc=es
objectClass: person
objectClass: inetOrgPerson
uid: ana
cn: Ana García
ou: ventas
mail: ana@tienda.es
memberOf: cn=empleados,ou=grupos,dc=tienda,dc=es

dn: uid=luis,ou=personas,dc=tienda,dc=es
objectClass: person
uid: luis
cn: Luis Pérez
ou: soporte
mail: luis@gmail.com
memberOf: cn=empleados,ou=grupos,dc=tienda,dc=es
memberOf: cn=admins,ou=grupos,dc=tienda,dc=es

dn: uid=admin,dc=tienda,dc=es
objectClass: account
uid: admin

dn: uid=eva,ou=personas,dc=tienda,dc=es
objectClass: person
uid: eva
cn: Eva Ruiz
ou: Ventas
mail: eva@tienda.es
---
(mail=*@tienda.es)
(|(ou=soporte)
(memberOf=cn=admins,ou=grupos,dc=tienda,dc=es))
(&(objectclass=PERSON)(!(ou=ventas)))
(cn=*ga*)
(telefono=*)
```

SALIDA ESPERADA

```
(mail=*@tienda.es) → ana, eva
(|(ou=soporte)
(memberOf=cn=admins,ou=grupos,dc=tienda,dc=es)) →
luis
(&(objectclass=PERSON)(!(ou=ventas))) → luis
(cn=*ga*) → ana
(telefono=*) → (ninguna)
```

Git, Documentación y CI/CD

21. La siguiente versión semántica

Fácil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Cada línea es «versión tipo», donde la versión es MAYOR.MENOR.PARCHE y el tipo es mayor, menor o patch. Escribe la siguiente versión: mayor sube el primero y pone los otros a 0; menor sube el segundo y pone a 0 el parche; patch sube el último. Si la versión no tiene el formato correcto o el tipo no es válido, escribe «Versión no válida».

Código de partida (bash)

```
#!/bin/bash
while read -r version tipo || [ -n "$version" ]; do
    # TODO
    :
done
```

Ejemplo: Los tres tipos

ENTRADA

```
1.4.2 patch
1.4.2 minor
1.4.2 major
```

SALIDA ESPERADA

```
1.4.3
1.5.0
2.0.0
```

Ejemplo: Entradas no válidas

ENTRADA

```
1.4 patch
v2.0.0 minor
2.0.0 hotfix
```

SALIDA ESPERADA

```
Versión no válida
Versión no válida
Versión no válida
```

22. Una fusión a tres bandas

Medio · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Así fusiona Git dos ramas: compara cada línea de la versión común (base) con la nuestra y la suya. La entrada son tres versiones del mismo fichero con el mismo número de líneas, separadas por «---»: base, nuestra y suya. Para cada línea: si nuestra y suya coinciden, se queda esa; si solo cambió una de las dos respecto a la base, gana la que cambió; si cambiaron las dos de forma distinta, hay conflicto. Las líneas en conflicto consecutivas forman un único bloque: «<<<<<< nuestra», nuestras líneas, «=====», sus líneas, «>>>>>> suya». Escribe el fichero fusionado y, al final, «Conflictos: N» (número de bloques).

Código de partida (bash)

```
#!/bin/bash
mapfile -t todo
base=(); nuestra=(); suya=(); parte=0
for l in "${todo[@]"; do
    if [ "$l" = "---" ]; then ((parte++)); continue; fi
    case $parte in 0) base+=("$l") ;; 1) nuestra+=("$l") ;; 2) suya+=("$l") ;; esac
done

# TODO: fusiona línea a línea
printf '%s\n' "${nuestra[@]}"
echo "Conflictos: 0"
```

Ejemplo: Cambios compatibles

ENTRADA

```
# Tienda
version=1.0
puerto=8080
debug=true
---
# Tienda
version=1.1
puerto=8080
debug=true
---
# Tienda
version=1.0
puerto=8080
debug=false
```

SALIDA ESPERADA

```
# Tienda
version=1.1
puerto=8080
debug=false
Conflictos: 0
```

Ejemplo: Un conflicto de dos líneas

ENTRADA

```
a=1
b=2
c=3
d=4
---
a=1
b=20
c=30
d=4
---
a=1
b=200
c=300
d=40
```

SALIDA ESPERADA

```
a=1
<<<<<<< nuestra
b=20
c=30
=====
b=200
c=300
>>>>>>> suya
d=40
Conflictos: 1
```

23. Changelog a partir de los commits

Difícil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

La entrada son mensajes de commit con el formato de Conventional Commits («tipo(ámbito)!: descripción», con ámbito y ! opcionales). Genera un changelog con las secciones «## Cambios incompatibles» (commits con !), «## Novedades» (feat) y «## Correcciones» (fix), en ese orden, cada una con líneas «- descripción» en el orden original y omitiendo las secciones vacías. Los demás tipos (docs, chore, test...) se ignoran salvo que lleven !. Termina con una línea en blanco y «Siguiente versión: mayor», «minor», «patch» o «ninguna».

Código de partida (bash)

```
#!/bin/bash
rotos=(); novedades=(); correcciones=()
while IFS= read -r linea || [ -n "$linea" ]; do
    # TODO: clasifica el commit
    :
done
# TODO: escribe el changelog
```

Ejemplo: Versión con novedades

ENTRADA

```
feat(api): añadir login con token
fix: corregir el total del carrito
docs: actualizar el README
feat(web): filtro por categoría
chore: subir dependencias
```

SALIDA ESPERADA

```
## Novedades
- añadir login con token
- filtro por categoría
## Correcciones
- corregir el total del carrito

Siguiente versión: minor
```

Ejemplo: Cambio incompatible

ENTRADA

```
fix(api): validar el email
refactor(api)!: renombrar /users a /usuarios
test: pruebas del carrito
```

SALIDA ESPERADA

```
## Cambios incompatibles
- renombrar /users a /usuarios
## Correcciones
- validar el email

Siguiente versión: mayor
```

Ejemplo: Solo mantenimiento

ENTRADA

```
docs: arreglar erratas
chore(ci): cachear dependencias
```

SALIDA ESPERADA

Siguiente versión: ninguna

24. Las métricas DORA de un equipo

Difícil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Las cuatro métricas DORA miden lo bien que un equipo entrega software. La entrada es un registro con líneas «despliegue FECHA HORA commit FECHA HORA resultado» (resultado ok o fallo; la fecha del commit es cuándo se escribió el cambio desplegado) y «restaurado FECHA HORA» (el servicio vuelve a funcionar tras un fallo), en orden cronológico. Calcula: «Despliegues: N en D días (X por semana)», con D los días naturales entre el primer y el último despliegue, ambos incluidos, y X con un decimal; «Tiempo de entrega (mediana): H h M min», la mediana en minutos de (despliegue – commit), y con un número par de valores, la media entera de los dos centrales; «Tasa de fallos: P %» con un decimal; y «Tiempo de recuperación (media): M min» (sin decimales), la media de los minutos entre cada despliegue fallido y el primer «restaurado» posterior, o «—» si no hubo fallos.

Código de partida (bash)

```
#!/bin/bash
export LC_ALL=C TZ=UTC
# TODO
cat > /dev/null
```

Ejemplo: Una semana de trabajo

ENTRADA

```
despliegue 2026-09-21 10:00 commit 2026-09-20
16:30 ok
despliegue 2026-09-22 12:00 commit 2026-09-22
09:00 ok
despliegue 2026-09-23 17:00 commit 2026-09-23
15:15 fallo
restaurado 2026-09-23 17:40
despliegue 2026-09-24 11:00 commit 2026-09-23
18:00 ok
despliegue 2026-09-27 09:30 commit 2026-09-25
09:30 ok
```

SALIDA ESPERADA

Despliegues: 5 en 7 días (5.0 por semana)
Tiempo de entrega (mediana): 17 h 0 min
Tasa de fallos: 20.0 %
Tiempo de recuperación (media): 40 min

Ejemplo: Sin fallos

ENTRADA

```
despliegue 2026-01-10 08:00 commit 2026-01-09
08:00 ok
despliegue 2026-01-10 20:00 commit 2026-01-10
19:00 ok
```

SALIDA ESPERADA

Despliegues: 2 en 1 días (14.0 por semana)
Tiempo de entrega (mediana): 12 h 30 min
Tasa de fallos: 0.0 %
Tiempo de recuperación (media): —

25. ¿Qué jobs ejecuta el pipeline?

Muy difícil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Las plataformas de CI deciden qué trabajos (jobs) lanzar según sus dependencias, la rama y los ficheros cambiados. La entrada tiene una línea por job, en un orden en el que las dependencias aparecen antes: «nombre [needs=a,b] [ramas=main,release/*] [rutas=src/*,Dockerfile]», una línea «---» y el contexto: «rama=...», «cambios=f1,f2...» y «fallan=j1,j2...» (jobs que fallarán si se ejecutan). Para cada job, en orden y comprobando en este orden: si alguna dependencia terminó en FALLO o cancelado → «cancelado (por dep)» (la primera en su lista); si alguna se omitió →

«omitido (depende de dep)»; si tiene ramas y ninguna coincide con la rama (con comodines * como los de la shell) → «omitido (rama)»; si tiene rutas y ningún fichero cambiado coincide con ninguna → «omitido (sin cambios)»; si no, se ejecuta: «FALLO» si está en fallan u «ok». Termina con «Resultado: fallido» si algún job falló o «Resultado: éxito».

Código de partida (bash)

```
#!/bin/bash
jobs=(); declare -A ctx
enjobs=1
while IFS= read -r l || [ -n "$l" ]; do
    [ -z "$l" ] && continue
    if [ "$l" = "---" ]; then enjobs=0; continue; fi
    if (( enjobs )); then jobs+=("$l"); else ctx[${l%*=}]=${l#*=}; fi
done

# TODO: decide cada job
for l in "${jobs[@]}"; do echo "${l%% *}": ok; done
echo "Resultado: éxito"
```

Ejemplo: Una rama de funcionalidad

ENTRADA

```
lint
pruebas needs=lint
imagen needs=pruebas ramas=main,release/*
rutas=src/*,Dockerfile,package.json
docs rutas=docs/*
desplegar_staging needs=imagen ramas=main
desplegar_prod needs=desplegar_staging
ramas=release/*
---
rama=feat/filtros
cambios=src/app.js,README.md
fallan=
```

SALIDA ESPERADA

```
lint: ok
pruebas: ok
imagen: omitido (rama)
docs: omitido (sin cambios)
desplegar_staging: omitido (depende de imagen)
desplegar_prod: omitido (depende de
desplegar_staging)
Resultado: éxito
```

Ejemplo: Main con un test roto

ENTRADA

```
lint
pruebas needs=lint
imagen needs=pruebas ramas=main,release/*
rutas=src/*,Dockerfile,package.json
docs rutas=docs/*
desplegar_staging needs=imagen ramas=main
desplegar_prod needs=desplegar_staging
ramas=release/*
---
rama=main
cambios=src/app.js,docs/guia.md
fallan=pruebas
```

SALIDA ESPERADA

```
lint: ok
pruebas: FALLO
imagen: cancelado (por pruebas)
docs: ok
desplegar_staging: cancelado (por imagen)
desplegar_prod: cancelado (por desplegar_staging)
Resultado: fallido
```

Ejercicios de Despliegue de Aplicaciones Web

26. Conflictos de puertos en Docker Compose

Medio · Ejercicios de Despliegue de Aplicaciones Web · apuntesdam.com/subject/despliegue/topic/ejercicios-despliegue

Cada línea describe una publicación de puertos de un servicio: «servicio puerto_host:puerto_contenedor». Escribe «CONFLICTO puerto: servicio1, servicio2» por cada puerto del host usado por más de un servicio (en orden numérico de puerto y con los servicios en el orden en que aparecen), y «PRIVILEGIADO servicio: puerto» por cada puerto del host menor que 1024, en el orden de la entrada. Si no hay problemas, escribe «Sin problemas».

Código de partida (bash)

```
#!/bin/bash
declare -A usos
problemas=0

# TODO
```

Ejemplo: Con conflictos

ENTRADA

```
web 8080:80
api 3000:3000
adminer 8080:8080
proxy 80:80
proxy 443:443
docs 3000:80
```

SALIDA ESPERADA

```
CONFLICTO 3000: api, docs
CONFLICTO 8080: web, adminer
PRIVILEGIADO proxy: 80
PRIVILEGIADO proxy: 443
```

Ejemplo: Sin problemas

ENTRADA

```
web 8080:80
api 3000:3000
```

SALIDA ESPERADA

```
Sin problemas
```

27. Informe de un pipeline de CI

Medio · Ejercicios de Despliegue de Aplicaciones Web · apuntesdam.com/subject/despliegue/topic/ejercicios-despliegue

Cada línea es un paso de un pipeline: «paso estado segundos», con estado ok, error o omitido. Escribe «Pasos: N (ok: X, error: Y, omitido: Z)», «Duración total: M min S s» (sumando solo los pasos no omitidos), «Paso más lento: nombre (N s)» entre los ejecutados y, si hay algún error, «Primer error: nombre»; si no, «Resultado: correcto».

Código de partida (bash)

```
#!/bin/bash
while read -r paso estado segundos || [ -n "$paso" ]; do
    # TODO
    :
done
```

Ejemplo: Pipeline con error

ENTRADA

```
checkout ok 3
instalar ok 48
lint ok 12
tests error 95
build omitido 0
desplegar omitido 0
```

SALIDA ESPERADA

```
Pasos: 6 (ok: 3, error: 1, omitido: 2)
Duración total: 2 min 38 s
Paso más lento: tests (95 s)
Primer error: tests
```

Ejemplo: Pipeline correcto

ENTRADA

```
checkout ok 2
tests ok 61
build ok 130
desplegar ok 40
```

SALIDA ESPERADA

```
Pasos: 4 (ok: 4, error: 0, omitido: 0)
Duración total: 3 min 53 s
Paso más lento: build (130 s)
Resultado: correcto
```

28. Rotación de copias de seguridad

Difícil · Ejercicios de Despliegue de Aplicaciones Web · apuntesdam.com/subject/despliegue/topic/ejercicios-despliegue

La primera línea es la fecha de hoy y las siguientes, nombres de copias «backup-AAAA-MM-DD.tar.gz» (desordenados). Aplica esta política: se conservan todas las copias de los últimos 7 días (hoy incluido, es decir, con 0 a 6 días de antigüedad); de las que tienen entre 7 y 30 días, solo las de domingo; y las de más de 30 días se borran. Escribe «CONSERVAR nombre» o «BORRAR nombre» ordenadas de la más reciente a la más antigua, y al final «Conservadas: N · Borradas: M». Los nombres que no siguen el formato se ignoran.

Código de partida (bash)

```
#!/bin/bash
read -r hoy

# TODO
```

Ejemplo: Un mes de copias

ENTRADA

```
2025-09-30
backup-2025-09-30.tar.gz
backup-2025-09-24.tar.gz
backup-2025-09-23.tar.gz
backup-2025-09-21.tar.gz
backup-2025-09-20.tar.gz
backup-2025-09-14.tar.gz
backup-2025-08-31.tar.gz
backup-2025-08-24.tar.gz
notas.txt
backup-2025-09-29.tar.gz
```

SALIDA ESPERADA

```
CONSERVAR backup-2025-09-30.tar.gz
CONSERVAR backup-2025-09-29.tar.gz
CONSERVAR backup-2025-09-24.tar.gz
BORRAR backup-2025-09-23.tar.gz
CONSERVAR backup-2025-09-21.tar.gz
BORRAR backup-2025-09-20.tar.gz
CONSERVAR backup-2025-09-14.tar.gz
CONSERVAR backup-2025-08-31.tar.gz
BORRAR backup-2025-08-24.tar.gz
Conservadas: 6 · Borradas: 3
```

Ejercicios largos

29. Informe de un registro de accesos web

Difícil · Bash · 75 minutos · apuntesdam.com/ejercicios/bash/informe-registro-accesos-web

Los servidores web como Apache y Nginx anotan cada petición en un registro de accesos (`access.log`). Cada línea dice quién pidió qué, cuándo y cómo respondió el servidor, en un formato estándar llamado Common Log Format.

El administrador de una web pequeña quiere un resumen diario sin instalar herramientas de análisis: un script de Bash que lea el registro por la entrada estándar (`./informe.sh < access.log`) y escriba las cifras importantes.

Un registro real puede tener líneas cortadas (por ejemplo, si el disco se llenó a mitad de una escritura). El script no debe contarlas como peticiones, pero sí decir cuántas había.

Requisitos

- Cada línea tiene la forma `IP - - [fecha zona] "MÉTODO ruta PROTOCOLO" código bytes` . Separada por espacios, la fecha ocupa dos campos y la petición tres; el código es el 9.º campo y los bytes el 10.º. Los bytes valen `-` cuando la respuesta no tiene cuerpo (por ejemplo, una redirección): cuentan como 0.
- Las líneas vacías se ignoran. Una línea no es válida si no llega al campo de los bytes o si el código no es un número de tres cifras.
- Si no hay ninguna petición válida, escribe solo `No hay peticiones válidas.`
- Escribe `Peticiones: N` , `Visitantes distintos: N` (el número de IP distintas), `Respuestas: 2xx A · 3xx B · 4xx C · 5xx D` (cuántas respuestas hay de cada familia, según la primera cifra del código) y `Datos enviados: N KB` (la suma de los bytes dividida entre 1024, sin decimales).

- Bajo **IPs con más peticiones:** , las 3 IP con más peticiones (o menos, si no hay tantas), de más a menos: el número en 3 caracteres alineado a la derecha, un espacio y la IP. A igual número, por orden de texto de la IP, carácter a carácter (por eso **10.0.0.10** va antes que **10.0.0.2**).
- **Página más pedida: ruta (N veces)** , con «vez» si es 1. Si hay empate, la primera en orden de texto.
- **Errores 404:** y debajo, con dos espacios delante, cada ruta distinta que haya respondido 404, en orden de texto. Si no hay ninguna, **Errores 404: ninguno** en una sola línea.
- Si ha habido líneas no válidas, termina con **Líneas no válidas: N** .

Formato de la entrada

- El registro de accesos por la entrada estándar, una petición por línea en Common Log Format.

Campos de una línea del registro

Campo	Ejemplo	Significado
1	192.168.1.10	IP del cliente
2 y 3	--	identidad y usuario (casi siempre vacíos)
4 y 5	[03/Oct/2026:10:15:32 +0200]	fecha, hora y zona horaria
6, 7 y 8	"GET /index.html HTTP/1.1"	método, ruta y protocolo
9	200	código de respuesta
10	5120	bytes enviados (- si no hay cuerpo)

Ejemplo: Diez peticiones

ENTRADA

```
192.168.1.10 - - [03/Oct/2026:10:15:32 +0200]
"GET /index.html HTTP/1.1" 200 5120
192.168.1.10 - - [03/Oct/2026:10:15:33 +0200]
"GET /css/estilos.css HTTP/1.1" 200 2048
10.0.0.7 - - [03/Oct/2026:10:16:01 +0200] "GET
/index.html HTTP/1.1" 200 5120
10.0.0.7 - - [03/Oct/2026:10:16:02 +0200] "GET
/favicon.ico HTTP/1.1" 404 512
172.16.0.3 - - [03/Oct/2026:10:17:45 +0200] "POST
/login HTTP/1.1" 302 -
172.16.0.3 - - [03/Oct/2026:10:17:46 +0200] "GET
/panel HTTP/1.1" 200 8192
192.168.1.10 - - [03/Oct/2026:10:18:10 +0200]
"GET /admin HTTP/1.1" 404 512
10.0.0.7 - - [03/Oct/2026:10:19:00 +0200] "GET
/index.html HTTP/1.1" 200 5120
192.168.1.10 - - [03/Oct/2026:10:20:15 +0200]
"GET /api/datos HTTP/1.1" 500 128
8.8.4.4 - - [03/Oct/2026:10:21:30 +0200] "GET
/index.html HTTP/1.1" 200 5120
```

SALIDA ESPERADA

```
Peticiones: 10
Visitantes distintos: 4
Respuestas: 2xx 6 · 3xx 1 · 4xx 2 · 5xx 1
Datos enviados: 31 KB
IPs con más peticiones:
  4 192.168.1.10
  3 10.0.0.7
  2 172.16.0.3
Página más pedida: /index.html (4 veces)
Errores 404:
  /admin
  /favicon.ico
```

Ejemplo: Empates y líneas rotas

ENTRADA

```
10.0.0.2 - - [04/Oct/2026:09:00:00 +0200] "GET /b
HTTP/1.1" 200 1024
10.0.0.10 - - [04/Oct/2026:09:00:01 +0200] "GET
/a HTTP/1.1" 200 1024
línea rota
10.0.0.2 - - [04/Oct/2026:09:00:02 +0200] "GET /a
HTTP/1.1" 301 -
10.0.0.10 - - [04/Oct/2026:09:00:03 +0200] "GET
/b HTTP/1.1" 200 2048

10.0.0.3 - - [04/Oct/2026:09:00:04 +0200] "GET /c
HTTP/1.1" 200 1024
```

SALIDA ESPERADA

```
Peticiones: 5
Visitantes distintos: 3
Respuestas: 2xx 4 · 3xx 1 · 4xx 0 · 5xx 0
Datos enviados: 5 KB
IPs con más peticiones:
  2 10.0.0.10
  2 10.0.0.2
  1 10.0.0.3
Página más pedida: /a (2 veces)
Errores 404: ninguno
Líneas no válidas: 1
```

Código de partida (bash)

```
#!/bin/bash
# Orden de texto byte a byte: así sort y [[ < ]] ordenan igual en cualquier máquina
export LC_ALL=C

declare -A por_ip por_ruta rutas_404
total=0
bytes_total=0
invalidas=0
c2=0 c3=0 c4=0 c5=0

while read -r ip _ _ _ metodo ruta _ codigo bytes _; do
    [ -z "$ip" ] && continue
    # TODO: valida la línea y actualiza los contadores
done

# TODO: escribe el informe
```

30. Analizador de la configuración de Nginx

Muy difícil · Bash · 90 minutos · apuntesdam.com/ejercicios/bash/analizador-de-configuracion-nginx

En un servidor con varias webs, la configuración de Nginx crece con el tiempo: un `server` para el dominio, otro para la API, otro que nadie recuerda... y los errores aparecen: un sitio que escucha en el 443 sin certificado (Nginx no arranca), un `autoindex on` olvidado que enseña los ficheros a cualquiera, o un dominio con HTTPS cuyo puerto 80 sigue sirviendo la web sin cifrar.

`nginx -t` comprueba la sintaxis, pero no esas buenas prácticas. Vas a escribir un script que lea un fichero de sitios (el que estaría en `/etc/nginx/sites-available/`), resuma cada bloque `server` y avise de los problemas.

El formato de Nginx es sencillo de leer línea a línea: directivas terminadas en `;` y bloques entre llaves (`server { ... }`), y dentro bloques `location ... { ... }`). Para saber dónde termina cada `server` hay que llevar la cuenta de las llaves abiertas.

Requisitos

- Se ignoran los comentarios (desde `#` hasta el final de la línea), los espacios de los extremos y las líneas vacías. Cada línea es `server {`, otra apertura de bloque (termina en `{`), un cierre `}` o una directiva terminada en `;`. Las líneas se numeran desde 1 contando todas.
- Errores, que detienen el script: `Error: llave de cierre sin abrir en la línea N`, `Error: bloque fuera de un server en la línea N`, `Error: directiva fuera de un server en la línea N` y, al final, `Error: llaves sin cerrar`. Si no hay ningún `server`, `No hay ningún bloque server`.
- De cada `server` se guardan (también desde dentro de sus `location`): los puertos de `listen` (el número tras los dos puntos, si los hay, y `ssl` si la línea lo lleva; sin repetir; 80 si no hay ningún `listen`), los nombres de `server_name`, el primer `root`, el primer `proxy_pass`, la URL del último `return` (lo que va tras el código), si tiene `ssl_certificate`, si tiene `ssl_certificate_key` y si tiene `autoindex on`.
- Por cada servidor: `Servidor N (líneas A-B): puerto P · nombres · acción`, con los puertos separados por `y` (443 `ssl` y 80), los nombres tal cual o (sin nombre), y como acción `redirige a URL`, `proxy a URL`, `sirve RUTA` o `no sirve nada`, por ese orden de preferencia.
- Avisos, primero los de cada servidor en orden (`Servidor N: usa ssl pero le falta ssl_certificate y ssl_certificate_key` —o solo el que falte—, `Servidor N: autoindex activado (muestra la lista de ficheros)`, `Servidor N: no tiene root, return ni proxy_pass`) y después, por orden alfabético, `dominio: aparece en K servidores del puerto P` (para cada dominio y puerto con más de un servidor) y `dominio: el puerto 80 no redirige a HTTPS` (para cada dominio que se sirve en el 443 sin un servidor en el 80 cuyo `return` vaya a `https://`).
- Escribe `Avisos:` y cada aviso con `-` delante, o `Sin avisos`.

Formato de la entrada

- Un fichero de configuración de sitios de Nginx.

Ejemplo: Un dominio con API

ENTRADA

```
# Sitio principal
server {
    listen 80;
    listen [::]:80;
    server_name ejemplo.com www.ejemplo.com;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name ejemplo.com www.ejemplo.com;
    ssl_certificate
/etc/letsencrypt/live/ejemplo.com/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/ejemplo.com/privkey.pem;
    root /var/www/ejemplo;

    location /descargas/ {
        autoindex on;    # se dejó activado tras
una prueba
    }
}

server {
    listen 443 ssl;
    server_name api.ejemplo.com;
    ssl_certificate /etc/ssl/api.crt;
    location / {
        proxy_pass http://127.0.0.1:3000;
    }
}
```

SALIDA ESPERADA

Servidor 1 (líneas 2-7): puerto 80 · ejemplo.com
www.ejemplo.com · redirige a
https://\$host\$request_uri
Servidor 2 (líneas 9-20): puerto 443 ssl ·
ejemplo.com www.ejemplo.com · sirve
/var/www/ejemplo
Servidor 3 (líneas 22-29): puerto 443 ssl ·
api.ejemplo.com · proxy a http://127.0.0.1:3000
Avisos:
- Servidor 2: autoindex activado (muestra la
lista de ficheros)
- Servidor 3: usa ssl pero le falta
ssl_certificate_key
- api.ejemplo.com: el puerto 80 no redirige a
HTTPS

Ejemplo: Servidores repetidos y uno vacío

ENTRADA

```
server {
    server_name intranet.local;
}

server {
    listen 8080;
    server_name intranet.local;
    root /srv/intranet;
}

server {
    listen 8080;
    server_name intranet.local;
    return 302 /mantenimiento.html;
}
```

SALIDA ESPERADA

Servidor 1 (líneas 1-3): puerto 80 ·
intranet.local · no sirve nada
Servidor 2 (líneas 4-8): puerto 8080 ·
intranet.local · sirve /srv/intranet
Servidor 3 (líneas 9-13): puerto 8080 ·
intranet.local · redirige a /mantenimiento.html
Avisos:
- Servidor 1: no tiene root, return ni
proxy_pass
- intranet.local: aparece en 2 servidores del
puerto 8080

Código de partida (bash)

```
#!/bin/bash
# Revisa un fichero de sitios de Nginx (como /etc/nginx/sites-available/web) y avisa de los fallos
típicos
export LC_ALL=C

n=0 profundidad=0 numero=0
declare -a inicio fin puertos nombres raiz retorno proxy cert clave autoindex

while IFS= read -r linea || [ -n "$linea" ]; do
    numero=$((numero + 1))
    # TODO: quita comentarios y espacios, sigue las llaves y guarda las directivas de cada server
done

# TODO: resumen de cada servidor y avisos
```

31. Despliegue por releases con enlace simbólico y vuelta atrás

Difícil · Bash · 70 minutos · apuntesdam.com/ejercicios/bash/despliegue-por-releases-con-rollback

Copiar los ficheros nuevos encima de los de la web en producción tiene un problema: durante unos segundos conviven ficheros viejos y nuevos, y si algo sale mal no hay forma rápida de volver atrás. Las herramientas de despliegue (Capistrano, Deployer, Envoyer) lo resuelven con releases: cada versión se instala en su propia carpeta (`releases/20261004-1000`) y el servidor web apunta a un enlace simbólico `current`.

Desplegar es preparar la carpeta nueva por completo y, solo al final, cambiar el enlace. Ese cambio se hace de forma atómica (se crea un enlace temporal y se renombra encima del antiguo con `mv -T`), así que ninguna petición ve una web a medias. Volver atrás es cambiar el enlace a la release anterior: un segundo, sin reinstalar nada.

El script recibe las releases que ya hay y una serie de órdenes, y escribe las órdenes de shell que ejecutaría. Las líneas informativas empiezan por `#`, así la salida se puede revisar y, después, ejecutar tal cual.

Requisitos

- La primera línea es `RELEASES` seguida de las carpetas que ya existen, de la más antigua a la más reciente (puede no haber ninguna); la más reciente es la actual. Si no empieza por `RELEASES`, escribe `La primera línea debe ser RELEASES r1 r2...` (puede estar vacía) y termina. Las órdenes pueden ir en minúsculas; las líneas vacías se ignoran; una orden desconocida escribe `# Orden desconocida: orden`.
- `DESPLIEGA AAAAMMDD-HHMM vX.Y.Z`: si el formato no es ese, `# Uso: DESPLIEGA AAAAMMDD-HHMM vX.Y.Z`; si la release ya existe, `# La release R ya existe`; si es anterior a la más reciente, `# La release R es anterior a ÚLTIMA`. Si todo va bien, escribe `# Desplegando vX.Y.Z en R` y las cinco órdenes de la tabla, la release pasa a ser la actual y, si hay más de 3, se escribe `rm -rf /var/www/tienda/releases/R` para borrar las más antiguas hasta dejar 3.
- `ROLLBACK`: si no hay actual o es la más antigua, `# No hay una release anterior a la que volver`; si no, `# Volviendo de ACTUAL a ANTERIOR`, la orden del enlace y la recarga de PHP-FPM, y la anterior pasa a ser la actual (se puede volver atrás varias veces seguidas).
- `ESTADO`: `# Releases (N):` y una línea por release de la más antigua a la más reciente, `# R` y, en la actual, `# R ← current`; o `# Sin releases`.
- Un despliegue después de un rollback despliega la versión nueva como más reciente (la release a la que se había vuelto deja de ser la actual).

Formato de la entrada

- Línea 1: `RELEASES` y las releases existentes. Resto: `DESPLIEGA fecha vX.Y.Z`, `ROLLBACK` o `ESTADO`.

Órdenes de un despliegue (R es la release, base `/var/www/tienda`)

Paso	Orden
Descargar la versión	<code>git clone --depth 1 --branch vX.Y.Z https://git.ejemplo.com/tienda.git /var/www/tienda/releases/R</code>
Configuración compartida	<code>ln -s /var/www/tienda/shared/.env /var/www/tienda/releases/R/.env</code>
Dependencias	<code>composer install --no-dev --optimize-autoloader --working-dir=/var/www/tienda/releases/R</code>
Cambio atómico	<code>ln -sfn /var/www/tienda/releases/R /var/www/tienda/current.tmp && mv -T /var/www/tienda/current.tmp /var/www/tienda/current</code>
Recargar PHP	<code>systemctl reload php8.3-fpm</code>

Ejemplo: Dos despliegues y una vuelta atrás

ENTRADA

```
RELEASES 20260901-1200 20260915-0930
ESTADO
DESPLIEGA 20261001-1800 v1.4.0
DESPLIEGA 20261004-1000 v1.5.0
ROLLBACK
ESTADO
```

SALIDA ESPERADA

```
# Releases (2):
# 20260901-1200
# 20260915-0930 ← current
# Desplegando v1.4.0 en 20261001-1800
git clone --depth 1 --branch v1.4.0
https://git.ejemplo.com/tienda.git
/var/www/tienda/releases/20261001-1800
ln -s /var/www/tienda/shared/.env
/var/www/tienda/releases/20261001-1800/.env
composer install --no-dev --optimize-autoloader -
-working-dir=/var/www/tienda/releases/20261001-
1800
ln -sfn /var/www/tienda/releases/20261001-1800
/var/www/tienda/current.tmp && mv -T
/var/www/tienda/current.tmp
/var/www/tienda/current
systemctl reload php8.3-fpm
# Desplegando v1.5.0 en 20261004-1000
git clone --depth 1 --branch v1.5.0
https://git.ejemplo.com/tienda.git
/var/www/tienda/releases/20261004-1000
ln -s /var/www/tienda/shared/.env
/var/www/tienda/releases/20261004-1000/.env
composer install --no-dev --optimize-autoloader -
-working-dir=/var/www/tienda/releases/20261004-
1000
ln -sfn /var/www/tienda/releases/20261004-1000
/var/www/tienda/current.tmp && mv -T
/var/www/tienda/current.tmp
/var/www/tienda/current
systemctl reload php8.3-fpm
rm -rf /var/www/tienda/releases/20260901-1200
# Volviendo de 20261004-1000 a 20261001-1800
ln -sfn /var/www/tienda/releases/20261001-1800
/var/www/tienda/current.tmp && mv -T
/var/www/tienda/current.tmp
/var/www/tienda/current
systemctl reload php8.3-fpm
# Releases (3):
# 20260915-0930
# 20261001-1800 ← current
# 20261004-1000
```

Ejemplo: Órdenes que no valen

ENTRADA

```
RELEASES 20261001-1800
DESPLIEGA 2026-10-04 v1.5
DESPLIEGA 20261001-1800 v1.5.0
DESPLIEGA 20260930-0800 v1.5.0
ROLLBACK
PUBLICA
```

SALIDA ESPERADA

```
# Uso: DESPLIEGA AAAAMMDD-HHMM vX.Y.Z
# La release 20261001-1800 ya existe
# La release 20260930-0800 es anterior a
20261001-1800
# No hay una release anterior a la que volver
# Orden desconocida: PUBLICA
```

Código de partida (bash)

```
#!/bin/bash
# Despliegue por releases (como Capistrano o Deployer): cada versión en su carpeta y un enlace «current»
# que se cambia de golpe. Modo simulación: escribe las órdenes en lugar de ejecutarlas.
export LC_ALL=C
BASE=/var/www/tienda
MAX_RELEASES=3

releases=()          # carpetas de releases, de la más antigua a la más reciente
actual=""

read -r palabra resto
# TODO: lee las releases existentes y atiende DESPLIEGA, ROLLBACK y ESTADO
```

Soluciones

1. ¿Cuántos nuevos tiene tu arquitectura?

Medio · Arquitecturas de Despliegue y la Nube · apuntesdam.com/subject/despliegue/topic/arquitecturas-despliegue

Solución (bash)

```
#!/bin/bash
export LC_ALL=C

awk '
$1 == "objetivo" { objetivo = $2; next }
$1 == "capa" {
    a = $3 / 100
    capa = 1 - (1 - a) ^ $4          # réplicas en paralelo: la capa cae si caen todas
    if (n++ == 0) total = capa; else total = total * capa  # capas en serie: se multiplican
    printf "%s: %d x %.2f %% → %.6f %%\n", $2, $4, $3, capa * 100
}
END {
    printf "Total: %.6f %%\n", total * 100
    printf "Caída máxima al mes: %.1f min\n", (1 - total) * 43200
    if (total * 100 >= objetivo) print "Cumple el objetivo"
    else printf "No cumple el objetivo del %.2f %%\n", objetivo
}'
```

La probabilidad de que caigan a la vez las n réplicas de una capa es $(1 - a)^n$, así que la capa funciona con probabilidad $1 - (1 - a)^n$. Las capas, en cambio, están en serie: el sistema solo funciona si funcionan todas, y su disponibilidad es el producto. Con tres servidores web al 99,5 % la capa supera el 99,9999 %, pero una sola base de datos sin réplica limita todo el sistema.

awk hace los cálculos con decimales (bash solo sabe de enteros) y `printf` da el formato; `%%` escribe el símbolo del porcentaje. `LC_ALL=C` garantiza que el separador decimal sea el punto.

2. Un balanceador ponderado con caídas

Difícil · Arquitecturas de Despliegue y la Nube · apuntesdam.com/subject/despliegue/topic/arquitecturas-despliegue

Solución (bash)

```
#!/bin/bash
declare -A peso cw activo cuenta
orden=()
while read -r a b && [ "$a" != "---" ]; do
    orden+=("$a"); peso[$a]=$b; cw[$a]=0; activo[$a]=1; cuenta[$a]=0
done

while read -r evento nombre || [ -n "$evento" ]; do
    case $evento in
        peticion)
            total=0; elegido=""
            for s in "${orden[@]"; do
                (( activo[$s] )) || continue
                (( cw[$s] += peso[$s], total += peso[$s] ))
                if [ -z "$elegido" ] || (( cw[$s] > cw[$elegido] )); then elegido=$s; fi
            done
            if [ -z "$elegido" ]; then
                echo "503 sin servidores"
                continue
            fi
            (( cw[$elegido] -= total, cuenta[$elegido] += 1 ))
            echo "→ $elegido"
            ;;
        caido) activo[$nombre]=0; cw[$nombre]=0; echo "$nombre fuera de servicio" ;;
        vuelve) activo[$nombre]=1; cw[$nombre]=0; echo "$nombre de vuelta" ;;
    esac
done

reparto=()
for s in "${orden[@]"; do reparto+=("$s=${cuenta[$s]}"); done
(IFS=,; echo "Reparto: ${reparto[*]}" | sed 's/,/, /g')
```

El algoritmo se escribe tal cual en tres pasos por petición: sumar a cada servidor activo su peso, elegir el de mayor contador (con `>` estricto, en empate gana el primero de la lista) y restarle la suma de los pesos. El efecto es que los servidores pesados reciben más peticiones, pero intercaladas (a a b a c a a) en lugar de en ráfagas (a a a a b c).

Los arrays asociativos de bash guardan el estado de cada servidor, y dentro de `(( ))` se pueden hacer varias operaciones aritméticas separadas por comas. Un servidor caído simplemente se salta en el bucle, y al volver empieza con su contador a 0.

3. Auditoría de un servidor recién instalado

Muy difícil · Arquitecturas de Despliegue y la Nube · apuntesdam.com/subject/despliegue/topic/arquitecturas-despliegue

Solución (bash)

```
#!/bin/bash
awk '
/^---$/ { bloque++; next }
bloque == 0 {                                # sshd_config
    if ($0 ~ /^[ \t]*(#|$)/) next
    clave = tolower($1)
    if (!(clave in conf)) conf[clave] = $2      # cuenta la primera aparición
    next
}
bloque == 1 && NF >= 3 {                      # puertos en escucha
    dir = $2; sub(/:[0-9]+$/, "", dir)
    puerto = $2; sub(/.*:/, "", puerto)
    if ((dir == "0.0.0.0" || dir == ":::") || dir == "") && !(puerto in programa) {
        programa[puerto] = $3
        lista[++n] = puerto + 0
    }
    next
}
bloque == 2 && $1 == "permitidos" {
    m = split($2, p, ",")
    for (i = 1; i <= m; i++) permitido[p[i] + 0] = 1
}
END {
    if (tolower(conf["permitrootlogin"]) == "yes") {
        print "ALTO: PermitRootLogin yes permite entrar como root con contraseña"; altos++
    }
    clave = "yes"                                # valor por defecto
    if ("passwordauthentication" in conf) clave = tolower(conf["passwordauthentication"])
    if (clave != "no") { print "ALTO: PasswordAuthentication permite contraseñas (usa solo claves)";
        altos++ }
    intentos = 6
    if ("maxauthtries" in conf) intentos = conf["maxauthtries"] + 0
    if (intentos > 4) { print "MEDIO: MaxAuthTries " intentos " (recomendado 4 o menos)"; medios++ }
    if (tolower(conf["x11forwarding"]) == "yes") { print "MEDIO: X11Forwarding activado sin necesidad";
        medios++ }

    for (i = 2; i <= n; i++) {                  # ordenar los puertos (inserción)
        x = lista[i]
        for (j = i - 1; j >= 1 && lista[j] > x; j--) lista[j + 1] = lista[j]
        lista[j + 1] = x
    }
    for (i = 1; i <= n; i++) {
        if (!(lista[i] in permitido)) {
            print "ALTO: puerto " lista[i] " (" programa[lista[i]] ") abierto a internet"; altos++
        }
    }
    if (altos + medios == 0) print "Resultado: sin problemas"
    else print "Resultado: " altos + 0 " altos, " medios + 0 " medios"
}'
```

Todo el análisis se hace con un único awk que lleva la cuenta del bloque en el que está (cada «---» lo incrementa). En sshd_config, la clave se pasa a minúsculas y solo se guarda la primera vez que aparece, que es como lo lee el propio sshd; las directivas que faltan toman su valor por defecto, que es precisamente lo peligroso (PasswordAuthentication yes, MaxAuthTries 6).

Para los puertos, `sub` separa la dirección del número; solo cuentan los que escuchan en todas las interfaces, y cada puerto se guarda una vez aunque aparezca en IPv4 e IPv6. mawk no tiene función de ordenación, así que los números se ordenan con un sencillo algoritmo de inserción antes de comprobarlos contra la lista de permitidos.

4. Generar un virtual host de Nginx

Fácil · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Solución (bash)

```
#!/bin/bash
read -r dominio raiz puerto

if [ -z "$dominio" ] || [ -z "$raiz" ]; then
    echo "Uso: dominio raíz [puerto]"
    exit 0
fi
echo "server {"
echo "    listen 80;"
echo "    server_name $dominio www.$dominio;"
if [ -n "$puerto" ]; then
    echo "        location / {"
    echo "            proxy_pass http://127.0.0.1:$puerto;"
    echo "        }"
else
    echo "        # sitio estático: Nginx sirve los ficheros"
    echo "        root $raiz;"
    echo "        index index.html;"
fi
echo "}"
```

read reparte la línea en tres variables; si falta alguna de las dos obligatorias se muestra el uso y se termina. El puerto es opcional y decide el tipo de bloque: sin él, Nginx sirve ficheros desde **root**; con él, reenvía las peticiones a la aplicación que escucha en ese puerto con **proxy_pass**.

Generar la configuración con un script en lugar de copiarla a mano evita errores tontos (un punto y coma que falta) cuando hay que dar de alta muchos sitios.

5. Estadísticas de un access.log

Medio · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Solución (bash)

```
#!/bin/bash
log=$(cat)

awk '
NF {
    n++
    peticiones[$1]++
    if ($9 ~ /^4/) e4++
    if ($9 ~ /^5/) e5++
}
END {
    for (ip in peticiones) {
        if (peticiones[ip] > max || (peticiones[ip] == max && ip < mejor)) { max = peticiones[ip]; mejor = ip }
    }
    printf "Peticiones: %d\nErrores 4xx: %d\nErrores 5xx: %d\nIP con más peticiones: %s (%d)\n", n, e4, e5, mejor, max
}' <<< "$log"
```

En el formato combinado de Nginx, separando por espacios, el primer campo es la IP y el noveno el código de estado (la fecha ocupa dos campos y la petición entre comillas, tres). **awk** cuenta las líneas, los códigos que empiezan por 4 y por 5, y las peticiones de cada IP en un array asociativo.

Al final se recorre el array buscando el máximo; en caso de empate gana la IP menor en orden alfabético, comparando las cadenas con **<**.

6. Vigilar la caducidad de los certificados

Medio · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Solución (bash)

```
#!/bin/bash
export TZ=UTC LC_ALL=C
read -r _ hoy

base=$(date -d "$hoy" +%s)
while read -r dominio fecha || [ -n "$dominio" ]; do
    [ -z "$dominio" ] && continue
    echo "$(( ($(date -d "$fecha" +%s) - base) / 86400 )) $dominio"    # días que quedan
done | sort -k1,1n -k2,2 | {
    criticos=0; avisos=0
    while read -r dias dominio; do
        if (( dias < 0 )); then
            echo "$dominio: CADUCADO hace $((-dias)) días"; criticos=$((criticos + 1))
        elif (( dias <= 7 )); then
            echo "$dominio: CRÍTICO: caduca en $dias días"; criticos=$((criticos + 1))
        elif (( dias <= 30 )); then
            echo "$dominio: AVISO: caduca en $dias días"; avisos=$((avisos + 1))
        else
            echo "$dominio: ok ($dias días)"
        fi
    done
    echo "Resumen: $criticos críticos, $avisos avisos"
}
```

`date -d fecha +%s` convierte una fecha en segundos desde 1970, y restando y dividiendo entre 86 400 salen los días que faltan (negativos si ya caducó). Se fija `TZ=UTC` para que ningún cambio de hora haga que un día dure 23 horas y la división entera se quede corta.

Cada dominio se escribe precedido de sus días y `sort -k1,1n -k2,2` ordena por días (numéricamente) y, a igualdad, por nombre. El segundo bucle está entre llaves dentro de la tubería para que los contadores sigan existiendo al escribir el resumen.

7. Cadenas y bucles de redirecciones

Difícil · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Solución (bash)

```
#!/bin/bash
declare -A regla
while read -r origen destino && [ "$origen" != "---" ]; do
    regla[$origen]=$destino
done

while read -r url || [ -n "$url" ]; do
    [ -z "$url" ] && continue
    camino=("$url"); actual=$url; bucle=0
    while [ -n "${regla[$actual]+x}" ]; do          # mientras haya una regla para la URL actual
        actual=${regla[$actual]}
        for v in "${camino[@]}"; do [ "$v" = "$actual" ] && bucle=1; done
        camino+=("$actual")
        (( bucle )) && break
    done
    saltos=$(( ${#camino[@]} - 1 ))
    if (( bucle )); then
        texto=${camino[0]}
        for (( i = 1; i < ${#camino[@]}; i++ )); do texto+=" → ${camino[i]}"; done
        echo "$url: BUCLE $texto"
    elif (( saltos == 0 )); then
        echo "$url: 200 sin redirección"
    elif (( saltos == 1 )); then
        echo "$url: 301 → $actual"
    else
        echo "$url: CADENA de $saltos saltos → $actual (redirige directamente)"
    fi
done
```

Para cada URL se siguen las reglas guardando el camino recorrido. Antes de añadir una URL nueva se comprueba si ya estaba en el camino: si es así, es un bucle y se para. `${regla[$actual]+x}` vale «x» solo si existe una regla para esa clave, lo que distingue «no hay regla» de «la regla está vacía».

Con el camino completo, el número de saltos decide el mensaje. Una cadena de varios saltos funciona, pero cada salto es una petición más para el navegador y para Google: lo correcto es que la URL antigua redirija directamente a la final.

8. ¿Qué location atiende la petición?

Muy difícil · Servidores Web: Apache y Nginx · apuntesdam.com/subject/despliegue/topic/servidores-web

Solución (bash)

```
#!/bin/bash
mods=(); pats=()
while read -r a b && [ "$a" != "---" ]; do
    if [ -n "$b" ]; then mods+=("$a"); pats+=("$b"); else mods+=(""); pats+=("$a"); fi
done

nombre() { if [ -n "${mods[$1]}" ]; then echo "${mods[$1]} ${pats[$1]}"; else echo "${pats[$1]}"; fi; }

while read -r uri || [ -n "$uri" ]; do
    [ -z "$uri" ] && continue
    elegida=-1
    # 1) Coincidencia exacta
    for i in "${!pats[@]}"; do
        if [ "${mods[$i]}" = "=" ] && [ "${pats[$i]}" = "$uri" ]; then elegida=$i; break; fi
    done
    if (( elegida < 0 )); then
        # 2) Prefijo más largo (normal o ^~)
        prefijo=-1
        for i in "${!pats[@]}"; do
            case ${mods[$i]} in ""|"^~") ;; *) continue ;; esac
            if [[ $uri == "${pats[$i]}"* ]] && { (( prefijo < 0 )) || (( ${#pats[$i]} > ${#pats[prefijo]} ) ) }; then prefijo=$i; fi
        done
        if (( prefijo >= 0 )) && [ "${mods[prefijo]}" = "^~" ]; then
            elegida=$prefijo
        else
            # 3) Expresiones regulares, en orden
            for i in "${!pats[@]}"; do
                case ${mods[$i]} in
                    "~") [[ $uri =~ ${pats[$i]} ]] && { elegida=$i; break; } ;;
                    "~*") shopt -s nocasematch
                        [[ $uri =~ ${pats[$i]} ]] && { shopt -u nocasematch; elegida=$i; break; }
                        shopt -u nocasematch ;;
                esac
            done
            # 4) Si ninguna coincide, el prefijo más largo
            (( elegida < 0 )) && elegida=$prefijo
        fi
    fi
    if (( elegida < 0 )); then echo "$uri → 404 (ninguna)"; else echo "$uri → $(nombre "$elegida")"; fi
done
```

El código sigue los cuatro pasos de Nginx en orden y se detiene en cuanto uno decide. Lo que más sorprende del algoritmo es el paso 2: el prefijo más largo no gana automáticamente, sino que se recuerda y solo se usa si ninguna expresión regular coincide; el modificador `^~` es justo la forma de decir «si este prefijo es el más largo, no mires las expresiones».

En bash, `[[ $uri == "$p"* ]]` comprueba si la URI empieza por el prefijo (las comillas hacen que el prefijo se tome literalmente) y `[[ $uri =~ $p ]]` aplica una expresión regular; para `~*` se activa `nocasematch` un momento, que hace que la comparación no distinga mayúsculas.

9. Estado de los servicios

Medio · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

Solución (bash)

```
#!/bin/bash
lentos=0; caidos=0
while read -r nombre codigo ms || [ -n "$nombre" ]; do
    [ -z "$nombre" ] && continue
    if [ "$codigo" != 200 ]; then
        estado="CAÍDO"; caidos=$((caidos + 1))
    elif (( ms >= 500 )); then
        estado="LENTO"; lentos=$((lentos + 1))
    else
        estado="OK"
    fi
    echo "$nombre: $estado"
done

if (( caidos > 0 )); then general="CAÍDO"
elif (( lentos > 0 )); then general="DEGRADADO"
else general="OK"; fi
echo "Estado general: $general"
```

Cada servicio se clasifica con una cadena de `if`: primero lo más grave (cualquier código distinto de 200 es una caída) y después la lentitud. Los contadores de caídos y lentos permiten decidir el estado general al final: basta con un servicio caído para que todo el sistema se considere caído.

`|| [ -n "$nombre" ]` hace que se procese también la última línea aunque no termine en salto de línea, un detalle típico al leer ficheros en bash.

10. Revisar el server.xml de Tomcat

Medio · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

Solución (bash)

```
#!/bin/bash
awk '
# Valor del atributo nombre="..." en la línea actual ("" si no está)
function attr(nombre, largo) {
    if (match($0, nombre "=\\\[^\"]*\\\"")) {
        largo = length(nombre) + 2
        return substr($0, RSTART + largo, RLENGTH - largo - 1)
    }
    return ""
}
/^[\t]*<!--/ { next } # comentarios
/<Server[ >]/ {
    puerto = attr("port"); orden = attr("shutdown")
    print "Apagado: puerto " puerto ", orden " orden
    if (puerto != "-1" && orden == "SHUTDOWN") avisos[++n] = "AVISO: puerto de apagado " puerto " con la
        orden por defecto"
}
/<Connector[ >]/ {
    puerto = attr("port"); protocolo = attr("protocol")
    if (protocolo == "") protocolo = "HTTP/1.1"
    tls = ""
    if (attr("SSLEnabled") == "true") tls = " (TLS)"
    print "Conector " puerto " " protocolo tls
    dir = attr("address")
    if (protocolo ~ /AJP/ && dir != "127.0.0.1" && dir != "::1" && dir != "localhost")
        avisos[++n] = "PELIGRO: conector AJP " puerto " accesible desde fuera (Ghostcat)"
}
/<Host[ >]/ {
    host = attr("name"); auto = attr("autoDeploy")
    if (auto == "") auto = "true"
    print "Host " host " → " attr("appBase") " (autoDeploy=" auto ")"
    if (auto == "true") avisos[++n] = "AVISO: autoDeploy activado en " host
}
END {
    for (i = 1; i <= n; i++) print avisos[i]
    print "Avisos: " n + 0
}'
```

Como cada elemento está en una línea, awk puede reconocerlos con un patrón (`/<Connector[ >]/`) y sacar sus atributos con la función `attr`, que busca `nombre="..."` con `match` y recorta el valor con `substr` usando `RSTART` y `RLENGTH`. Los valores por defecto (HTTP/1.1, autoDeploy=true) se aplican cuando el atributo no aparece.

Los avisos se guardan en un array mientras se lee y se escriben al final, después del inventario. Los tres son fallos reales de seguridad: la orden de apagado por defecto en un puerto abierto, el conector AJP expuesto (la vulnerabilidad Ghostcat de 2020) y el despliegue automático de cualquier WAR que se copie en el servidor.

11. Despliegue atómico con releases y vuelta atrás

Difícil · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

Solución (bash)

```
#!/bin/bash
base=/tmp/app
rm -rf "$base" && mkdir -p "$base/releases"
orden=() # releases en el orden en que se desplegaron

estado() {
    if [ ! -L "$base/current" ]; then echo "Sin desplegar"; return; fi
    local actual
    actual=$(readlink "$base/current")
    echo "current → ${actual##*/} (releases: ${orden[*]})"
}

while read -r accion version || [ -n "$accion" ]; do
    case $accion in
        desplegar)
            if [ -d "$base/releases/$version" ]; then echo "Error: $version ya existe"; continue; fi
            mkdir "$base/releases/$version"
            ln -sf "$base/releases/$version" "$base/current" # el cambio es atómico
            orden+=("$version")
            while (( ${#orden[@]} > 3 )); do # solo las 3 más recientes
                rm -rf "$base/releases/${orden[0]}"
                orden=( "${orden[@]:1}" )
            done
            estado
            ;;
        rollback)
            actual=$(readlink "$base/current" 2>/dev/null); actual=${actual##*/}
            anterior=""
            for i in "${!orden[@]"; do
                if [ "${orden[i]}" = "$actual" ] && (( i > 0 )); then anterior=${orden[i-1]}; fi
            done
            if [ -z "$anterior" ]; then echo "Error: no hay versión anterior"; continue; fi
            ln -sf "$base/releases/$anterior" "$base/current"
            estado
            ;;
    esac
done
estado) estado ;;
```

La web siempre sirve lo que hay en `current`, que es solo un enlace simbólico. `ln -sf` lo cambia de destino en una operación, así que no hay ningún momento en que la web vea una versión a medias; y volver atrás es apuntar el enlace a la carpeta anterior, que sigue ahí. Es el método de Capistrano, Deployer y muchos scripts de despliegue.

El array `orden` recuerda en qué orden se desplegaron las versiones: sirve para saber cuál es la anterior a la actual y cuáles son las más antiguas al limpiar. Como la nueva siempre es la última, borrar desde el principio nunca elimina la actual.

12. Despliegue canary: ¿promocionar o revertir?

Muy difícil · Servidores de Aplicaciones · apuntesdam.com/subject/despliegue/topic/servidores-aplicaciones

Solución (bash)

```
#!/bin/bash
export LC_ALL=C

awk '
BEGIN { split("5 25 50 100", etapa, " "); e = 1 }
NF >= 7 {                                     # $1 $2 $3 = estable . $5 $6 $7 = canario
    n++
    pct = etapa[e]
    if ($5 < 50) { printf "%d: %d %% → datos insuficientes (%d peticiones)\n", n, pct, $5; next }
    te = $2 / $1 * 100
    tc = $6 / $5 * 100
    if (tc - te > 0.5) {
        printf "%d: %d %% → REVERTIDA (errores %.2f %% frente a %.2f %%)\n", n, pct, tc, te; fin = 1;
        exit
    }
    if ($7 > $3 * 1.2) {
        printf "%d: %d %% → REVERTIDA (latencia %d ms frente a %d ms)\n", n, pct, $7, $3; fin = 1; exit
    }
    if (pct == 100) { printf "%d: 100 %% → ok, PROMOCIONADA\n", n; fin = 1; exit }
    e++
    printf "%d: %d %% → ok, pasa al %d %%\n", n, pct, etapa[e]
}
END { if (!fin) printf "Sin decisión: se mantiene el %d %%\n", etapa[e] }'
```

Las etapas del despliegue están en un array y un índice indica en cuál estamos; cada evaluación favorable lo avanza. Las comprobaciones van en el orden del enunciado y la primera que decide termina: sin datos suficientes no se puede comparar, y un empeoramiento claro de errores o de latencia revierte la versión.

`exit` en `awk` salta directamente al bloque `END`, que solo escribe «Sin decisión» si no se tomó ninguna decisión antes (la variable `fin`). Las tasas de error se comparan en puntos porcentuales y la latencia en proporción (más de un 20 % peor), que son los criterios habituales de herramientas como Argo Rollouts o Flagger.

13. Revisar un Dockerfile

Medio · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

Solución (bash)

```
#!/bin/bash
dockerfile=$(cat)
aviso=()

sin_version=0; apt_sucio=0; copia_pronto=0; usuario=0; copiado_todo=0
instalar='npm (install|ci)|composer install|pip install'
while IFS= read -r l; do
    if [[ $l =~ ^FROM[:space:]]+([^\[:space:]]+) ]]; then
        imagen=${BASH_REMATCH[1]}
        [[ $imagen != *: || $imagen == *:latest ]] && sin_version=1
    elif [[ $l =~ ^RUN ]]; then
        [[ $l == *"apt-get install"* && $l != *"rm -rf /var/lib/apt/lists"* ]] && apt_sucio=1
        [[ $l =~ $instalar ]] && (( copiado_todo )) && copia_pronto=1
    elif [[ $l =~ ^COPY[:space:]]+([^\[:space:]]+) ]]; then
        copiado_todo=1
    elif [[ $l =~ ^USER[:space:]]+([^\[:space:]]+) ]]; then
        usuario=1
    fi
done <<< "$dockerfile"

(( sin_version )) && aviso+=("Imagen base sin versión concreta")
(( apt_sucio )) && aviso+=("apt-get install sin limpiar la caché")
(( copia_pronto )) && aviso+=("Se copia todo el código antes de instalar las dependencias")
(( usuario )) || aviso+=("El contenedor se ejecuta como root")

if (( ${#aviso[@]} )); then printf '%s\n' "${aviso[@]}"; else echo "Sin avisos"; fi
```

El Dockerfile se recorre línea a línea y cada instrucción activa una bandera; los avisos se escriben al final en el orden fijo del enunciado. `[[ ... =~ ... ]]` aplica expresiones regulares y `BASH_REMATCH` guarda lo capturado, como el nombre de la imagen del FROM.

El aviso más útil es el tercero: si `COPY . .` va antes de instalar las dependencias, cualquier cambio en el código invalida la caché de esa capa y Docker vuelve a instalar todas las dependencias en cada construcción. Para detectarlo basta con recordar si ya se había copiado todo cuando aparece el RUN que instala.

14. Puertos de un docker compose

Medio · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

Solución (bash)

```
#!/bin/bash
declare -A usado
errores=0

valido() { [[ $1 =~ ^[0-9]+$ ]] && (( 10#$1 >= 1 && 10#$1 <= 65535 )); }

while IFS= read -r linea || [ -n "$linea" ]; do
    [ -z "$linea" ] && continue
    servicio=${linea%:*}
    resto=${linea#*:}
    for p in $resto; do
        if [[ $p != *.* ]]; then
            echo "$servicio: $p solo interno (no se publica)"
            continue
        fi
        host=${p%:*}; contenedor=${p#*:}
        if ! valido "$host" || ! valido "$contenedor"; then
            echo "$servicio: $p ERROR puerto no válido"; errores=$((errores + 1)); continue
        fi
        if [ -n "${usado[$host]}" ]; then
            echo "$servicio: $p ERROR el puerto $host ya lo publica ${usado[$host]}"; errores=$((errores + 1)); continue
        fi
        usado[$host]=$servicio
        privilegiado=""
        (( host < 1024 )) && privilegiado=" (privilegiado)"
        echo "$servicio: $host → $contenedor$privilegiado"
    done
done
echo "Publicados: ${#usado[@]}, errores: $errores"
```

Las expansiones de parámetros separan las partes sin programas externos: `${p%:*}` quita todo desde los dos puntos (queda el puerto del anfitrión) y `${p#*:}` quita hasta ellos (queda el del contenedor). Un puerto sin dos puntos no se publica: solo lo ven los demás contenedores de la misma red.

El array asociativo `usado` recuerda qué servicio publicó cada puerto del anfitrión, lo que permite detectar el conflicto (dos contenedores no pueden escuchar en el mismo puerto de la máquina) y nombrar al culpable; su tamaño es el número de puertos publicados. La función `valido` usa `10#` para que un número con ceros a la izquierda no se lea en octal.

15. Orden de arranque según depends_on

Difícil · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

Solución (bash)

```
#!/bin/bash
declare -A dep
orden=()
while IFS=: read -r s d || [ -n "$s" ]; do
    [ -z "$s" ] && continue
    s=$(echo $s); dep[$s]=$(echo $d); orden+=("$s")
done

# 1) Dependencias que no existen
for s in "${orden[@]"; do
    for d in "${dep[$s]"; do
        if [ -z "${dep[$d]+x}" ]; then echo "Error: $s depende de $d, que no existe"; exit 0; fi
    done
done

# 2) Ciclos: búsqueda en profundidad en orden alfabético
declare -A estado      # 1 = en el camino actual, 2 = terminado
pila=()
dfs() {
    local s=$1 d v texto en=0
    estado[$s]=1; pila+=("$s")
    for d in $(printf '%s\n' "${dep[$s]} | LC_ALL=C sort); do
        if [ "${estado[$d]}" = 1 ]; then                # vuelve a un servicio del camino: ciclo
            texto=""
            for v in "${pila[@]"; do [ "$v" = "$d" ] && en=1; (( en )) && texto+="$v → "; done
            echo "Ciclo: $texto$d"; exit 0
        fi
        [ -z "${estado[$d]}" ] && dfs "$d"
    done
    estado[$s]=2; unset 'pila[${#pila[@]}-1]'
}
ordenados=$(printf '%s\n' "${orden[@]} | LC_ALL=C sort)
for s in $ordenados; do [ -z "${estado[$s]}" ] && dfs "$s"; done

# 3) Rondas: arrancan a la vez los que ya tienen todas sus dependencias
declare -A arrancado
hechos=0; ronda=0
while (( hechos < ${#orden[@]} )); do
    lista=()
    for s in $ordenados; do
        [ -n "${arrancado[$s]}" ] && continue
        listo=1
        for d in "${dep[$s]"; do [ -z "${arrancado[$d]}" ] && listo=0; done
        (( listo )) && lista+=("$s")
    done
    for s in "${lista[@]"; do arrancado[$s]=1; done
    hechos=$((hechos + ${#lista[@]}); ronda=$((ronda + 1))
    echo "Ronda $ronda: ${lista[*]}"
done
```

Antes de arrancar nada hay que comprobar que el grafo de dependencias tiene sentido. Las dependencias inexistentes se detectan con `${dep[$d]+x}`, que distingue un servicio sin dependencias (existe con valor vacío) de uno que no existe. Los ciclos se buscan con una búsqueda en profundidad que marca los servicios del camino actual: si se llega a uno marcado, el trozo de pila desde él es el ciclo.

Las rondas son la versión «por niveles» del orden topológico: en cada vuelta se recogen todos los servicios no arrancados cuyas dependencias ya están en marcha, y se marcan juntos al final de la ronda (si se marcaran uno a uno, un servicio podría arrancar en la misma ronda que su dependencia).

16. ¿Qué capas se reconstruyen?

Muy difícil · Contenedores con Docker · apuntesdam.com/subject/despliegue/topic/docker

Solución (bash)

```
#!/bin/bash
mapfile -t todo
bloque=0; v1=(); v2=(); cambiados=()
for l in "${todo[@]}"; do
    if [ "$l" = "---" ]; then ((bloque++)); continue; fi
    [ -z "$l" ] && continue
    case $bloque in 0) v1+=("$l") ;; 1) v2+=("$l") ;; 2) cambiados+=("$l") ;; esac
done

# ¿Afecta el fichero f a la fuente src de un COPY?
afecta() {
    local f=$1 src=$2
    [ "$src" = "." ] && return 0
    [[ $src == "*" ]] && { [[ $f == $src ]]; return; } # patrón con comodín
    [ "$f" = "$src" ] || [[ $f == "${src%}/"/* ]] # fichero o carpeta
}

roto=0; reutilizadas=0
for i in "${!v2[@]}"; do
    instr=${v2[i]}; motivo=""
    if (( roto )); then
        motivo="capa anterior reconstruida"
    elif (( i >= ${#v1[@]} )) || [ "$instr" != "${v1[i]}" ]; then
        motivo="instrucción cambiada"
    else
        read -ra partes <<< "$instr"
        if [ "${partes[0]}" = COPY ] || [ "${partes[0]}" = ADD ]; then
            args=(); otra_etapa=0
            for p in "${partes[@]:1}"; do
                case $p in --from=*) otra_etapa=1 ;; --*) ;; *) args+=("$p") ;; esac
            done
            if (( ! otra_etapa )); then
                fuentes=("${args[@]:0:${#args[@]}-1}") # el último es el destino
                for f in "${cambiados[@]}"; do
                    for src in "${fuentes[@]}"; do
                        if afecta "$f" "$src"; then motivo="cambió $f"; break 2; fi
                    done
                done
            fi
        fi
    fi
    if [ -n "$motivo" ]; then
        roto=1
        echo "$((i + 1)) $instr → RECONSTRUIR ($motivo)"
    else
        reutilizadas=$((reutilizadas + 1))
        echo "$((i + 1)) $instr → caché"
    fi
done
echo "Capas reutilizadas: $reutilizadas de ${#v2[@]}"
```

La caché de Docker funciona como una cadena: cada capa depende de la anterior, así que en cuanto una se reconstruye (la bandera `roto`) todas las siguientes también. Mientras no ocurra, una capa se reutiliza si la instrucción es idéntica y, si es un COPY o ADD, si no ha cambiado ninguno de los ficheros que copia.

La función `afecta` resuelve qué ficheros cubre cada fuente: «.» los cubre todos, un patrón como `package*.json` se compara con `[[ $f == $src ]]` (sin comillas, para que el `*` actúe de comodín) y una carpeta cubre todo lo que empieza por su ruta. Un `COPY --from` copia de otra etapa y no depende de los ficheros del proyecto.

Los ejemplos muestran por qué importa el orden del Dockerfile: copiando primero solo `package*.json`, un cambio en el código no obliga a reinstalar las dependencias.

17. Leer una zona DNS

Medio · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

Solución (bash)

```
#!/bin/bash
awk '
function completo(x) {           # nombre absoluto
    if (x == "@") return origen
    if (x ~ /\.$/) return x
    return x "." origen
}
$1 == "$ORIGIN" { origen = $2; next }
NF == 0 || /^[ \t]*;/ || /^$/ { next } # vacías, comentarios y otras directivas
{
    i = 2
    if ($2 == "IN") i = 3           # la clase es opcional
    tipo = $i
    nombre = completo($1)
    if (tipo == "A" || tipo == "AAAA") print nombre " " tipo " " $(i + 1)
    else if (tipo == "CNAME") print nombre " CNAME " completo($(i + 1))
    else if (tipo == "MX") print nombre " MX " $(i + 1) " " completo($(i + 2))
    else next
    n++
}
END { print "Registros: " n + 0 }'
```

La clave del formato de zona es la regla de los nombres: «@» es el origen, un nombre acabado en punto es absoluto y cualquier otro es relativo y se completa añadiéndole el origen. La función `completo` aplica esa regla y se usa tanto para el nombre del registro como para los destinos de CNAME y MX.

El tipo está en el tercer campo si la línea lleva la clase IN y en el segundo si no, así que se calcula su posición `i` y se accede con `$i`. Los registros que no interesan se saltan con `next`, que también evita contarlos.

18. Revisar una zona DNS antes de publicarla

Medio · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

Solución (bash)

```
#!/bin/bash
awk '
function completo(x) {
    if (x == "@") return origen
    if (x ~ /\./) return x
    return x "." origen
}
$1 == "$ORIGIN" { origen = $2; dominio = substr(origen, 1, length(origen) - 1); next }
NF == 0 || /^[ \t]*$/ || /^$/ { next }
{
    i = 2
    if ($2 == "IN") i = 3
    nombre = completo($1); tipo = $i
    if (!(nombre in visto)) { visto[nombre] = 1; nombres[++nn] = nombre }
    if (tipo == "CNAME") cname[nombre] = 1
    else otros[nombre] = 1
    destino = ""
    if (tipo == "CNAME" || tipo == "NS") destino = $(i + 1)
    if (tipo == "MX") destino = $(i + 2)
    if (destino != "") { n++; rtipo[n] = tipo; rnombre[n] = nombre; rdestino[n] = destino }
}
END {
    for (k = 1; k <= nn; k++) {
        x = nombres[k]
        if (!(x in cname)) continue
        if (x == origen) { print "ERROR: CNAME en el ápice (" x ")"; errores++ }
        else if (x in otros) { print "ERROR: " x " tiene un CNAME y otros registros"; errores++ }
    }
    for (k = 1; k <= n; k++) {
        if (rtipo[k] != "MX") continue
        d = completo(rdestino[k])
        if (d in cname) { print "ERROR: el MX de " rnombre[k] " apunta a " d ", que es un CNAME";
            errores++ }
    }
    for (k = 1; k <= n; k++) {
        d = rdestino[k]
        if (d ~ /\./) continue
        if (d == dominio || substr(d, length(d) - length(dominio)) == "." dominio) {
            print "AVISO: falta el punto final en «" d "» (se leería " d "." origen ")"; avisos++
        }
    }
    print "Errores: " errores + 0 ", avisos: " avisos + 0
}'
```

Primero se lee toda la zona y se anota, por cada nombre (en orden de aparición), si tiene un CNAME y si tiene otros registros; además se guardan los registros que tienen destino (CNAME, MX y NS). Las tres comprobaciones se hacen en el bloque END, una detrás de otra, porque necesitan conocer la zona completa: un MX puede apuntar a un nombre definido más abajo.

Las reglas vienen del propio DNS: un nombre con CNAME no puede tener ningún otro registro (y el ápice siempre tiene SOA y NS, por eso allí nunca puede haber CNAME), y un MX debe apuntar a un nombre con dirección. El aviso del punto final detecta el error más típico de las zonas: escribir «tienda.es» sin punto hace que se lea como relativo y se convierta en «tienda.es.tienda.es.».

19. Un resolvidor DNS iterativo con caché

Difícil · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

Solución (bash)

```
#!/bin/bash
declare -A registro          # "servidor|nombre|tipo" -> "valor ttl"
declare -A delegaciones      # servidor -> lista de "dominio|servidor_hijo|ttl"
while read -r srv nom tipo val ttl && [ "$srv" != "---" ]; do
    if [ "$tipo" = "NS" ]; then delegaciones[$srv]+="$nom|$val|$ttl "; else
        registro["$srv|$nom|$tipo"]="$val $ttl"; fi
done

declare -A cacheA cacheNS    # nombre -> "ip caduca" · dominio -> "servidor caduca"
contiene() { [[ $2 == "$1" || $2 == *."$1" ]]; } # ¿el dominio $1 contiene al nombre $2?

total=0
while read -r t nom || [ -n "$t" ]; do
    [ -z "$t" ] && continue
    if [ -n "${cacheA[$nom]}" ]; then
        read -r ip caduca <<< "${cacheA[$nom]}"
        if (( t < caduca )); then echo "$t $nom: $ip (caché)"; continue; fi
    fi
    # Servidor de partida: la delegación en caché más específica que contenga el nombre
    srv=raiz; largo=0
    for dom in "${!cacheNS[@]"; do
        read -r hijo caduca <<< "${cacheNS[$dom]}"
        if (( t < caduca )) && contiene "$dom" "$nom" && (( ${#dom} > largo )); then srv=$hijo;
            largo=${#dom}; fi
    done
    consultas=0; resultado=""
    while true; do
        consultas=$((consultas + 1))
        if [ -n "${registro["$srv|$nom|A"]}" ]; then
            read -r ip ttl <<< "${registro["$srv|$nom|A"]}"
            cacheA[$nom]="$ip $((t + ttl))"
            resultado=$ip; break
        fi
        siguiente=""; largo=0
        for e in "${delegaciones[$srv]"; do
            IFS='|' read -r dom hijo ttl <<< "$e"
            if contiene "$dom" "$nom" && (( ${#dom} > largo )); then siguiente=$hijo; mejor=$dom;
                mejorTtl=$ttl; largo=${#dom}; fi
        done
        [ -z "$siguiente" ] && break # nadie sabe más: no existe
        cacheNS[$mejor]="$siguiente $((t + mejorTtl))"
        srv=$siguiente
    done
    total=$((total + consultas))
    if [ -n "$resultado" ]; then echo "$t $nom: $resultado ($consultas consultas)"; else echo "$t $nom:
        no existe ($consultas consultas)"; fi
done
echo "Consultas totales: $total"
```

El resolvidor tiene dos cachés: la de respuestas (nombre → IP) y la de delegaciones (dominio → servidor que lo conoce), cada una con su instante de caducidad (instante de la consulta más TTL). Si la respuesta está en caché y vigente, no se pregunta a nadie; si no, se empieza por la delegación vigente más específica que se conozca, y solo si no hay ninguna por la raíz. Por eso la segunda consulta de un mismo dominio cuesta una sola pregunta.

La resolución iterativa es un bucle: cada servidor responde con la dirección (y se termina) o con una delegación hacia un servidor más cercano al nombre (y se le pregunta a él). La función `contiene` decide si un dominio abarca un nombre: «tienda.es.» contiene a «www.tienda.es.» porque este acaba en «.tienda.es.».

Con los TTL se ve el compromiso del DNS: valores largos ahorran consultas, pero un cambio de IP tarda más en llegar a todo el mundo.

20. Evaluar filtros de búsqueda LDAP

Muy difícil · FTP, DNS y Servicios de Directorio · apuntesdam.com/subject/despliegue/topic/servicios-red

Solución (bash)

```
#!/bin/bash
awk '
# Expresión regular equivalente a un valor LDAP con comodines *
function patron(v, r, i, ch) {
    r = "^"
    for (i = 1; i <= length(v); i++) {
        ch = substr(v, i, 1)
        if (ch == "*") r = r ".*"
        else if (index("\\.[](){}|", ch)) r = r "\\" ch
        else r = r ch
    }
    return r "$"
}

# Evalúa el filtro que empieza en F[pos] para la entrada e (descenso recursivo)
function evalua(e, c, r, sr, fin, item, eq, at, val, re, k) {
    pos++ # salta el (
    c = substr(F, pos, 1)
    if (c == "&" || c == "|") {
        pos++
        r = (c == "&")
        while (substr(F, pos, 1) == "(") {
            sr = evalua(e) # se evalúan todos para avanzar pos
            if (c == "&" && !sr) r = 0
            if (c == "|" && sr) r = 1
        }
        pos++ # salta el )
        return r
    }
    if (c == "!") {
        pos++
        r = !evalua(e)
        pos++
        return r
    }
    fin = index(substr(F, pos), ")")
    item = substr(F, pos, fin - 1)
    pos += fin
    eq = index(item, "=")
    at = tolower(substr(item, 1, eq - 1))
    val = tolower(substr(item, eq + 1))
    if (val == "") return cnt[e, at] > 0 # presencia del atributo
    re = patron(val)
    for (k = 1; k <= cnt[e, at]; k++) if (tolower(vals[e, at, k]) ~ re) return 1
    return 0
}

!filtros && $0 == "---" { filtros = 1; next }
!filtros {
    if ($0 ~ /^[ \t]*$/) { nueva = 1; next }
    if (nueva || ne == 0) { ne++; nueva = 0 }
    p = index($0, ":")
    at = tolower(substr($0, 1, p - 1))
    v = substr($0, p + 1); sub(/^\+/, "", v)
    cnt[ne, at]++
    vals[ne, at, cnt[ne, at]] = v
    if (at == "uid" && !(ne in uid)) uid[ne] = v
    next
}

NF {
    F = $0
    lista = ""
    for (x = 1; x <= ne; x++) {
        pos = 1
        if (evalua(x)) {
            if (lista != "") lista = lista ", "
            lista = lista uid[x]
        }
    }
}
```

```

}
if (lista == "") lista = "(ninguna)"
print F " → " lista
}'

```

Los filtros LDAP tienen una gramática recursiva, así que se evalúan con un analizador de descenso recursivo: la función `evalua` lee un paréntesis y, según su primer carácter, combina subfiltros con Y (&), O (|) o NO (!), o compara un atributo con un valor. Una variable global `pos` marca por dónde va la lectura; por eso se evalúan todos los subfiltros aunque el resultado ya se sepa, para que `pos` avance hasta el final del grupo.

Cada entrada guarda todos los valores de cada atributo (un atributo puede repetirse) y una condición se cumple si la cumple alguno. Los comodines se traducen a una expresión regular: el valor se escapa carácter a carácter, `*` pasa a ser `.*` y se ancla con `^` y `$`; `attr=*` es el caso especial de «el atributo existe».

En awk las variables son globales salvo que se declaren como parámetros extra de la función: por eso `evalua` lista sus variables locales en la cabecera, imprescindible en una función recursiva.

21. La siguiente versión semántica

Fácil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Solución (bash)

```

#!/bin/bash
while read -r version tipo || [ -n "$version" ]; do
    [ -z "$version" ] && continue
    if [[ ! $version =~ ^([0-9]+)\.([0-9]+)\.([0-9]+)$ ]]; then
        echo "Versión no válida"
        continue
    fi
    mayor=$((10#${BASH_REMATCH[1]})); menor=$((10#${BASH_REMATCH[2]})); parche=$((10#${BASH_REMATCH[3]}))
    case $tipo in
        major) echo "$((mayor + 1)).0.0" ;;
        minor) echo "$mayor.$((menor + 1)).0" ;;
        patch) echo "$mayor.$menor.$((parche + 1))" ;;
        *) echo "Versión no válida" ;;
    esac
done

```

La expresión regular comprueba el formato y, con sus tres grupos, separa los números a la vez: después están en `BASH_REMATCH[1]`, `[2]` y `[3]`. El prefijo `10#` obliga a leerlos en base 10 (un «09» se tomaría como octal y daría error).

Cada tipo de cambio sube su número y pone a cero los de su derecha: es la regla del versionado semántico, que permite saber por el número si una actualización puede romper algo.

22. Una fusión a tres bandas

Medio · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Solución (bash)

```
#!/bin/bash
mapfile -t todo
base=(); nuestra=(); suya=(); parte=0
for l in "${todo[@]"; do
    if [ "$l" = "---" ]; then ((parte++)); continue; fi
    case $parte in 0) base+=("$l") ;; 1) nuestra+=("$l") ;; 2) suya+=("$l") ;; esac
done

salida=(); conflictos=0; bloque_n=(); bloque_s=()
cerrar() {
    # vuelca el bloque de conflicto pendiente, si lo hay
    (( ${#bloque_n[@]} )) || return 0
    salida+=("<<<<<< nuestra" "${bloque_n[@]}" "=====" "${bloque_s[@]}" ">>>>>> suya")
    conflictos=$((conflictos + 1))
    bloque_n=(); bloque_s=()
}
for i in "${!base[@]"; do
    b=${base[i]}; n=${nuestra[i]}; s=${suya[i]}
    if [ "$n" = "$s" ]; then cerrar; salida+=("$n")           # coinciden
    elif [ "$n" = "$b" ]; then cerrar; salida+=("$s")       # solo cambió la suya
    elif [ "$s" = "$b" ]; then cerrar; salida+=("$n")       # solo cambió la nuestra
    else bloque_n+=("$n"); bloque_s+=("$s")                 # cambiaron las dos: conflicto
    fi
done
cerrar
printf '%s\n' "${salida[@]}"
echo "Conflictos: $conflictos"
```

La fusión a tres bandas compara cada versión con la base común, no entre sí: si una de las dos ramas no ha tocado una línea, gana la que la cambió, sin conflicto. Solo cuando las dos la cambian de forma distinta Git no puede decidir y lo marca con `<<<<<<`, `=====` y `>>>>>>`.

Las líneas en conflicto seguidas se acumulan en dos arrays y la función `cerrar` las vuelca como un único bloque en cuanto aparece una línea sin conflicto (o al final). Por eso un cambio de dos líneas seguidas da un solo conflicto, igual que en Git.

23. Changelog a partir de los commits

Difícil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Solución (bash)

```
#!/bin/bash
rotos=(); novedades=(); correcciones=()
formato='^[a-z]+(\([^)]*\))?(!)?:(.*)$'
while IFS= read -r linea || [ -n "$linea" ]; do
    [[ $linea =~ $formato ]] || continue          # no sigue la convención
    tipo=${BASH_REMATCH[1]}; rompe=${BASH_REMATCH[3]}; descripcion=${BASH_REMATCH[4]}
    if [ -n "$rompe" ]; then rotos+=("$descripcion")
    elif [ "$tipo" = feat ]; then novedades+=("$descripcion")
    elif [ "$tipo" = fix ]; then correcciones+=("$descripcion")
    fi
done

seccion() {                                     # título y elementos; nada si está vacía
    local titulo=$1; shift
    (( $# )) || return 0
    echo "## $titulo"
    printf -- '- %s\n' "$@"
}

seccion "Cambios incompatibles" "${rotos[@]}"
seccion "Novedades" "${novedades[@]}"
seccion "Correcciones" "${correcciones[@]}"

if (( ${#rotos[@]} )); then siguiente=mayor
elif (( ${#novedades[@]} )); then siguiente=minor
elif (( ${#correcciones[@]} )); then siguiente=patch
else siguiente=ninguna; fi
echo
echo "Siguiente versión: $siguiente"
```

Una expresión regular, guardada en una variable para no tener que escapar los espacios, descompone cada commit en tipo, ámbito opcional, «!» opcional y descripción. Cada descripción va a uno de tres arrays según la regla de Conventional Commits: el «!» manda sobre el tipo.

La función `seccion` recibe el título y los elementos como argumentos y no escribe nada si no hay ninguno, así las tres secciones se generan igual. La siguiente versión sale del array más importante que tenga algo: incompatible → mayor, novedad → minor, corrección → patch.

24. Las métricas DORA de un equipo

Difícil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Solución (bash)

```
#!/bin/bash
export LC_ALL=C TZ=UTC

minutos() { echo $(( $(date -d "$1 $2" +%s) / 60 )); }

n=0; fallos=0; primero=""; ultimo=""
entregas=(); pendientes=(); recuperaciones=()
while read -r tipo f h _ cf ch resultado || [ -n "$tipo" ]; do
    case $tipo in
        despliegue)
            t=$(minutos "$f" "$h")
            n=$((n + 1)); [ -z "$primero" ] && primero=$f; ultimo=$f
            entregas+=($(t - $(minutos "$cf" "$ch"))) # del commit al despliegue
            if [ "$resultado" = fallo ]; then fallos=$((fallos + 1)); pendientes+=("$t"); fi
            ;;
        restaurado)
            t=$(minutos "$f" "$h")
            for p in "${pendientes[@]"; do recuperaciones+=($(t - p)); done
            pendientes=()
            ;;
    esac
done

dias=$(( $(date -d "$ultimo" +%s) - $(date -d "$primero" +%s) ) / 86400 + 1 )
mapfile -t orden < <(printf '%s\n' "${entregas[@]}") | sort -n
m=${#orden[@]}
if (( m % 2 )); then mediana=${orden[m / 2]}; else mediana=$(( (orden[m / 2 - 1] + orden[m / 2]) / 2 ));
fi

awk -v n="$n" -v dias="$dias" -v med="$mediana" -v fallos="$fallos" -v rec="${recuperaciones[*]}" 'BEGIN
{
    printf "Despliegues: %d en %d días (%.1f por semana)\n", n, dias, n / dias * 7
    printf "Tiempo de entrega (mediana): %d h %d min\n", int(med / 60), med % 60
    printf "Tasa de fallos: %.1f %%\n", fallos / n * 100
    k = split(rec, r, " ")
    if (k == 0) { print "Tiempo de recuperación (media): -"; exit }
    for (i = 1; i <= k; i++) s += r[i]
    printf "Tiempo de recuperación (media): %.0f min\n", s / k
}'
```

Todas las métricas se basan en restar instantes, así que cada fecha se convierte en minutos desde 1970 con `date -d ... +%s` (en UTC, para que no influyan los cambios de hora). Mientras se lee el registro se guardan el tiempo de entrega de cada despliegue y los despliegues fallidos que aún no se han restaurado; al llegar un «restaurado», cada uno de ellos aporta su tiempo de recuperación.

La mediana se calcula ordenando los tiempos con `sort -n`: es el central o, con una cantidad par, la media de los dos centrales. Se usa la mediana y no la media porque un solo cambio que tardó una semana en desplegarse no debería dar una imagen falsa del equipo. Los cálculos con decimales se hacen en un `awk` final, ya que `bash` solo trabaja con enteros.

25. ¿Qué jobs ejecuta el pipeline?

Muy difícil · Git, Documentación y CI/CD · apuntesdam.com/subject/despliegue/topic/ci-cd

Solución (bash)

```
#!/bin/bash
set -f                                     # que los * de las rutas no se expandan como ficheros
jobs=(); declare -A ctx
enjobs=1
while IFS= read -r l || [ -n "$l" ]; do
    [ -z "$l" ] && continue
    if [ "$l" = "---" ]; then enjobs=0; continue; fi
    if (( enjobs )); then jobs+=("$l"); else ctx[${l%*=}]=${l#*=}; fi
done

IFS=, read -ra cambios <<< "${ctx[cambios]}"
declare -A res                             # job -> ok | FALLO | cancelado | omitido
fallido=0
coincide() { local x=$1 p; shift; for p in "$@"; do [[ $x == $p ]] && return 0; done; return 1; }

for l in "${jobs[@]"; do
    read -r nombre opciones <<< "$l"
    needs=""; ramas=""; rutas=""
    for o in $opciones; do
        case $o in needs=*) needs=${o#needs=} ;; ramas=*) ramas=${o#ramas=} ;; rutas=*) rutas=${o#rutas=}
        ;; esac
    done
    IFS=, read -ra deps <<< "$needs"
    estado=""
    for d in "${deps[@]"; do
        case ${res[$d]} in FALLO|cancelado) estado="cancelado (por $d)"; break ;; esac
    done
    if [ -z "$estado" ]; then
        for d in "${deps[@]"; do [ "${res[$d]}" = omitido ] && { estado="omitido (depende de $d)";
            break; }; done
    fi
    if [ -z "$estado" ] && [ -n "$ramas" ]; then
        IFS=, read -ra pr <<< "$ramas"
        coincide "${ctx[rama]}" "${pr[@]}" || estado="omitido (rama)"
    fi
    if [ -z "$estado" ] && [ -n "$rutas" ]; then
        IFS=, read -ra pr <<< "$rutas"
        alguno=0
        for f in "${cambios[@]"; do coincide "$f" "${pr[@]}" && alguno=1; done
        (( alguno )) || estado="omitido (sin cambios)"
    fi
    if [ -z "$estado" ]; then
        if coincide "$nombre" ${ctx[fallan]//,/ }; then estado="FALLO"; fallido=1; else estado="ok"; fi
    fi
    res[$nombre]="${estado%% *}"
    echo "$nombre: $estado"
done
if (( fallido )); then echo "Resultado: fallido"; else echo "Resultado: éxito"; fi
```

Como los jobs vienen ordenados de forma que sus dependencias aparecen antes, basta con recorrerlos una vez guardando el resultado de cada uno en un array asociativo. Las condiciones se comprueban en el orden del enunciado y la primera que decide fija el estado; un job cancelado arrastra a los que dependen de él, igual que uno omitido.

La función `coincide` compara un valor con una lista de patrones usando `[[ $x == $p ]]` sin comillas en el patrón, que es lo que hace que `release/*` o `src/*` funcionen como los comodines de la shell. Al principio se desactiva la expansión de nombres de fichero con `set -f`, para que esos asteriscos no se sustituyan por ficheros reales al partir las opciones.

26. Conflictos de puertos en Docker Compose

Medio · Ejercicios de Despliegue de Aplicaciones Web · apuntesdam.com/subject/despliegue/topic/ejercicios-despliegue

Solución (bash)

```
#!/bin/bash
declare -A usos
problemas=0

privilegiados=()
while read -r servicio mapeo || [ -n "$servicio" ]; do
    [ -z "$servicio" ] && continue
    host=${mapeo%:*}
    if [ -z "${usos[$host]}" ]; then usos[$host]=$servicio; else usos[$host]+=", $servicio"; fi
    (( host < 1024 )) && privilegiados+=("PRIVILEGIADO $servicio: $host")
done

for host in $(printf '%s\n' "${!usos[@]}" | sort -n); do
    if [[ ${usos[$host]} == *,* ]]; then # más de un servicio en el mismo puerto
        echo "CONFLICTO $host: ${usos[$host]}"
        problemas=$((problemas + 1))
    fi
done
for p in "${privilegiados[@]}"; do
    echo "$p"
    problemas=$((problemas + 1))
done
(( problemas )) || echo "Sin problemas"
```

El array asociativo `usos` acumula, para cada puerto del anfitrión, los servicios que lo publican, separados por comas y en el orden de la entrada; si al final un puerto tiene alguna coma, hay conflicto. Las claves de un array asociativo no tienen orden, así que se listan con `${!usos[@]}` y se ordenan con `sort -n`.

Los puertos privilegiados se guardan aparte, en el orden en que aparecen, porque el enunciado pide ese orden y no el numérico.

27. Informe de un pipeline de CI

Medio · Ejercicios de Despliegue de Aplicaciones Web · apuntesdam.com/subject/despliegue/topic/ejercicios-despliegue

Solución (bash)

```
#!/bin/bash
n=0; ok=0; error=0; omitido=0; total=0; lento=""; maximo=-1; primer_error=""
while read -r paso estado segundos || [ -n "$paso" ]; do
    [ -z "$paso" ] && continue
    n=$((n + 1))
    case $estado in
        ok) ok=$((ok + 1)) ;;
        error) error=$((error + 1)); [ -z "$primer_error" ] && primer_error=$paso ;;
        omitido) omitido=$((omitido + 1)); continue ;; # no cuenta en el tiempo
    esac
    total=$((total + segundos))
    if (( segundos > maximo )); then maximo=$segundos; lento=$paso; fi
done

echo "Pasos: $n (ok: $ok, error: $error, omitido: $omitido)"
echo "Duración total: $((total / 60)) min $((total % 60)) s"
echo "Paso más lento: $lento ($maximo s)"
if [ -n "$primer_error" ]; then echo "Primer error: $primer_error"; else echo "Resultado: correcto"; fi
```

Un solo recorrido actualiza todos los contadores; los pasos omitidos se cuentan y después `continue` los salta, así no suman tiempo ni pueden ser el más lento. El primer error se guarda solo si todavía no había ninguno.

La división entera y el resto (`/` y `%`) convierten los segundos totales en minutos y segundos.

28. Rotación de copias de seguridad

Difícil · Ejercicios de Despliegue de Aplicaciones Web · apuntesdam.com/subject/despliegue/topic/ejercicios-despliegue

Solución (bash)

```
#!/bin/bash
export TZ=UTC LC_ALL=C
read -r hoy

h=$(date -d "$hoy" +%s)
conservadas=0; borradas=0
while read -r f; do
    fecha=${f:7:10}
    t=$(date -d "$fecha" +%s 2>/dev/null) || continue      # fecha imposible: se ignora
    dias=$(( (h - t) / 86400 ))
    dia_semana=$(date -d "$fecha" +%u)                    # 7 = domingo
    if (( dias <= 6 )) || { (( dias <= 30 )) && [ "$dia_semana" = 7 ]; }; then
        echo "CONSERVAR $f"; conservadas=$((conservadas + 1))
    else
        echo "BORRAR $f"; borradas=$((borradas + 1))
    fi
done < <(grep -E '^backup-[0-9]{4}-[0-9]{2}-[0-9]{2}\.tar\.gz$' | sort -r)
echo "Conservadas: $conservadas · Borradas: $borradas"
```

`grep -E` se queda solo con los nombres que siguen el formato y `sort -r` los ordena del más reciente al más antiguo (con fechas AAAA-MM-DD el orden alfabético coincide con el cronológico). El bucle lee de `< <(…)` en lugar de una tubería para que los contadores no se pierdan al terminar.

La antigüedad sale de restar las fechas en segundos y dividir entre 86 400, y `date +%u` da el día de la semana (7 es domingo). La política es una rotación típica: copias diarias de la última semana y semanales del último mes, que permite volver atrás bastante tiempo sin guardar 30 copias.

29. Informe de un registro de accesos web

Difícil · Bash · 75 minutos · apuntesdam.com/ejercicios/bash/informe-registro-accesos-web

Solución (bash)

```
#!/bin/bash
# Orden de texto byte a byte: así sort y [[ < ]] ordenan igual en cualquier máquina
export LC_ALL=C

declare -A por_ip por_ruta rutas_404
total=0
bytes_total=0
invalidas=0
c2=0 c3=0 c4=0 c5=0

while read -r ip _ _ _ metodo ruta _ codigo bytes _; do
    [ -z "$ip" ] && continue
    if [ -z "$bytes" ] || ! [[ $codigo =~ ^[0-9]{3}$ ]]; then
        invalidas=$((invalidas + 1))
        continue
    fi
    total=$((total + 1))
    por_ip[$ip]=$(( ${por_ip[$ip]:-0} + 1 ))
    por_ruta[$ruta]=$(( ${por_ruta[$ruta]:-0} + 1 ))
    # Una respuesta sin cuerpo trae «-» en lugar del tamaño
    [ "$bytes" != "-" ] && bytes_total=$((bytes_total + bytes))
    case ${codigo:0:1} in
        2) c2=$((c2 + 1)) ;;
        3) c3=$((c3 + 1)) ;;
        4) c4=$((c4 + 1)) ;;
        5) c5=$((c5 + 1)) ;;
    esac
    [ "$codigo" = "404" ] && rutas_404[$ruta]=1
done

if [ "$total" -eq 0 ]; then
    echo "No hay peticiones válidas."
    exit 0
fi

echo "Peticiones: $total"
echo "Visitantes distintos: ${#por_ip[@]}"
echo "Respuestas: 2xx $c2 · 3xx $c3 · 4xx $c4 · 5xx $c5"
echo "Datos enviados: $((bytes_total / 1024)) KB"

echo "IPs con más peticiones:"
for ip in "${!por_ip[@]}"; do
    echo "${por_ip[$ip]} $ip"
done | sort -k1,1nr -k2,2 | head -n 3 | while read -r n ip; do
    printf "%3d %s\n" "$n" "$ip"
done

mejor=""
for ruta in "${!por_ruta[@]}"; do
    n=${por_ruta[$ruta]}
    if [ -z "$mejor" ] || [ "$n" -gt "${por_ruta[$mejor]}" ] || { [ "$n" -eq "${por_ruta[$mejor]}" ] && [ $ruta < $mejor ] }; then
        mejor=$ruta
    fi
done

veces=${por_ruta[$mejor]}
[ "$veces" -eq 1 ] && palabra="vez" || palabra="veces"
echo "Página más pedida: $mejor ($veces $palabra)"

if [ ${#rutas_404[@]} -eq 0 ]; then
    echo "Errores 404: ninguno"
else
    echo "Errores 404:"
    printf "%s\n" "${!rutas_404[@]}" | sort | while read -r ruta; do
        echo " $ruta"
    done
done
```

```
fi  
  
[ "$invalidas" -gt 0 ] && echo "Líneas no válidas: $invalidas"  
exit 0
```

Todo el recuento se hace en una sola pasada: el bucle `while read` valida la línea y actualiza los contadores, y el informe se escribe al final con lo acumulado.

El bucle lee directamente de la entrada del script (`done` sin tubería delante). Si se escribiera `cat fichero | while read ...`, el bucle correría en una subshell y los contadores se perderían al terminar: es uno de los errores más típicos en Bash.

Los arrays asociativos hacen de diccionario: por IP, por ruta y, como conjunto, las rutas con 404. Ordenar sí se delega en `sort`, que lo hace mejor que cualquier bucle; la tubería del ranking sí corre en subshell, pero ahí solo se escribe.

`LC_ALL=C` fija el orden byte a byte. Sin él, `sort` usaría las reglas del idioma del sistema y el mismo script podría ordenar distinto en dos máquinas.

30. Analizador de la configuración de Nginx

Muy difícil · Bash · 90 minutos · apuntesdam.com/ejercicios/bash/analizador-de-configuracion-nginx

Solución (bash)

```
#!/bin/bash
# Revisa un fichero de sitios de Nginx (como /etc/nginx/sites-available/web) y avisa de los fallos
típicos
export LC_ALL=C

n=0 profundidad=0 numero=0
declare -a inicio fin puertos nombres raiz retorno proxy cert clave autoindex

while IFS= read -r linea || [ -n "$linea" ]; do
    numero=$((numero + 1))
    linea=${linea%%#*}                                # quita los comentarios
    linea="${linea%%[![:space:]]*}"                  # y los espacios de los extremos
    linea="${linea%%[![:space:]]*}"
    [ -z "$linea" ] && continue

    if [[ $linea =~ ^server[![:space:]]*\{$ ] && [ "$profundidad" -eq 0 ]; then
        n=$((n + 1)); inicio[$n]=$numero; profundidad=1
        continue
    fi
    if [[ $linea == "{" ]]; then
        # un bloque location o similar
        if [ "$profundidad" -eq 0 ]; then echo "Error: bloque fuera de un server en la línea $numero"; exit
        0; fi
        profundidad=$((profundidad + 1))
        continue
    fi
    if [ "$linea" = "}" ]]; then
        if [ "$profundidad" -eq 0 ]; then echo "Error: llave de cierre sin abrir en la línea $numero"; exit
        0; fi
        profundidad=$((profundidad - 1))
        [ "$profundidad" -eq 0 ] && fin[$n]=$numero
        continue
    fi
    if [ "$profundidad" -eq 0 ]; then echo "Error: directiva fuera de un server en la línea $numero"; exit
    0; fi

    linea=${linea%;}
    read -r directiva valor <<< "$linea"
    case $directiva in
        listen)
            puerto=${valor%*}; puerto=${puerto##*:}      # «[:]:443 ssl» → 443
            [[ $valor == *"ssl"* ] && puerto+=" ssl"
            [[ " ${puertos[$n]} " == "$puerto " ] || puertos[$n]="${puertos[$n]:+$puertos[$n]} $puerto"
            ;;
        server_name) nombres[$n]=$valor ;;
        root) [ -z "${raiz[$n]}" ] && raiz[$n]=$valor ;;
        return) retorno[$n]="${valor##* }" ;;
        proxy_pass) [ -z "${proxy[$n]}" ] && proxy[$n]=$valor ;;
        ssl_certificate) cert[$n]=1 ;;
        ssl_certificate_key) clave[$n]=1 ;;
        autoindex) [ "$valor" = "on" ] && autoindex[$n]=1 ;;
    esac
done

if [ "$profundidad" -ne 0 ]; then echo "Error: llaves sin cerrar"; exit 0; fi
if [ "$n" -eq 0 ]; then echo "No hay ningún bloque server"; exit 0; fi

avisos=()
declare -A servidores_de      # «nombre puerto» → número de servidores
declare -A redirige           # nombre → 1 si su servidor del 80 redirige a https
declare -A en443
for i in $(seq 1 "$n"); do
    [ -z "${puertos[$i]}" ] && puertos[$i]="80"
    if [ -n "${retorno[$i]}" ]; then hace="redirige a ${retorno[$i]}"
    elif [ -n "${proxy[$i]}" ]; then hace="proxy a ${proxy[$i]}"
    elif [ -n "${raiz[$i]}" ]; then hace="sirve ${raiz[$i]}"
    else hace="no sirve nada"; fi
done
```

```

echo "Servidor $i (líneas ${inicio[$i]}-${fin[$i]}): puerto ${puertos[$i]// / y } · ${nombres[$i]}:-(sin
nombre)} · $hace" | sed 's/ y ssl/ ssl/g'

if [[ " ${puertos[$i]}" == *" ssl"* ]]; then
    faltan=""
    [ -z "${cert[$i]}" ] && faltan="ssl_certificate"
    [ -z "${clave[$i]}" ] && faltan="${faltan:+$faltan y }ssl_certificate_key"
    [ -n "$faltan" ] && avisos+=("Servidor $i: usa ssl pero le falta $faltan")
fi
[ -n "${autoindex[$i]}" ] && avisos+=("Servidor $i: autoindex activado (muestra la lista de ficheros)")
[ "$hace" = "no sirve nada" ] && avisos+=("Servidor $i: no tiene root, return ni proxy_pass")

for nombre in ${nombres[$i]}; do
    for p in ${puertos[$i]// ssl/}; do
        servidores_de["$nombre $p"]=$(( ${servidores_de["$nombre $p"]:-0} + 1 ))
        if [ "$p" = "80" ] && [[ ${retorno[$i]} == https://* ]]; then redirige[$nombre]=1; fi
        [ "$p" = "443" ] && en443[$nombre]=1
    done
done
done

# Avisos por nombre de dominio, en orden alfabético
mapfile -t claves < <(printf "%s\n" "${!servidores_de[@]}" | sort)
for k in "${claves[@]"; do
    [ -z "$k" ] && continue
    [ "${servidores_de[$k]}" -gt 1 ] && avisos+=("${k%*}: aparece en ${servidores_de[$k]} servidores del
puerto ${k#* }")
done
mapfile -t seguros < <(printf "%s\n" "${!en443[@]}" | sort)
for nombre in "${seguros[@]"; do
    [ -z "$nombre" ] && continue
    [ -z "${redirige[$nombre]}" ] && avisos+=("$nombre: el puerto 80 no redirige a HTTPS")
done

if [ ${#avisos[@]} -eq 0 ]; then
    echo "Sin avisos"
else
    echo "Avisos:"
    for a in "${avisos[@]"; do echo "  - $a"; done
fi

```

El script es un analizador con estado: la profundidad de llaves dice si una línea pertenece a un servidor, a uno de sus `location` o a nada. Es la misma técnica que el conversor CSV con comillas, aplicada a un formato con bloques anidados.

Guardar los datos en arrays paralelos indexados por el número de servidor convierte el fichero en una pequeña tabla. Con ella, el resumen de cada servidor es un recorrido y los avisos son preguntas sobre la tabla.

Los avisos de dominios necesitan una segunda pasada que cruce información de varios servidores (el 80 de un dominio y su 443 suelen estar en bloques distintos). Por eso se recogen primero todos los datos y se analizan después.

En un servidor real, el script se ejecutaría antes de `nginx -t && systemctl reload nginx`, por ejemplo en un pipeline de despliegue: los fallos de sintaxis los encuentra Nginx; los de criterio, este script.

31. Despliegue por releases con enlace simbólico y vuelta atrás

Difícil · Bash · 70 minutos · apuntesdam.com/ejercicios/bash/despliegue-por-releases-con-rollback

Solución (bash)

```
#!/bin/bash
# Despliegue por releases (como Capistrano o Deployer): cada versión en su carpeta y un enlace «current»
# que se cambia de golpe. Modo simulación: escribe las órdenes en lugar de ejecutarlas.
export LC_ALL=C
BASE=/var/www/tienda
MAX_RELEASES=3

releases=()          # carpetas de releases, de la más antigua a la más reciente
actual=""

read -r palabra resto
if [ "$palabra" != "RELEASES" ]; then
    echo "La primera línea debe ser RELEASES r1 r2... (puede estar vacía)"
    exit 0
fi
for r in $resto; do releases+=("$r"); done
[ ${#releases[@]} -gt 0 ] && actual=${releases[-1]}

indice_de() {
    # posición de una release en el array, o -1
    local i
    for i in "${!releases[@]}"; do [ "${releases[$i]}" = "$1" ] && { echo "$i"; return; }; done
    echo -1
}

while read -r orden a b; do
    [ -z "$orden" ] && continue
    case ${orden^^} in
        DESPLIEGA)
            if [[ ! $a =~ ^[0-9]{8}-[0-9]{4}$ ]] || [[ ! $b =~ ^v[0-9]+\.[0-9]+\.[0-9]+$ ]]; then
                echo "# Uso: DESPLIEGA AAAAMMDD-HHMM vX.Y.Z"; continue
            fi
            if [ "$(indice_de "$a")" -ge 0 ]; then echo "# La release $a ya existe"; continue; fi
            if [ ${#releases[@]} -gt 0 ] && [ $a < ${releases[-1]} ]; then echo "# La release $a es anterior a ${releases[-1]}"; continue; fi
            echo "# Desplegando $b en $a"
            echo "git clone --depth 1 --branch $b https://git.ejemplo.com/tienda.git $BASE/releases/$a"
            echo "ln -s $BASE/shared/.env $BASE/releases/$a/.env"
            echo "composer install --no-dev --optimize-autoloader --working-dir=$BASE/releases/$a"
            # El cambio de versión es atómico: se crea un enlace temporal y se renombra sobre «current»
            echo "ln -sf $BASE/releases/$a $BASE/current.tmp && mv -T $BASE/current.tmp $BASE/current"
            echo "systemctl reload php8.3-fpm"
            releases+=("$a")
            actual=$a
            # Solo se conservan las últimas releases, sin borrar nunca la actual
            while [ ${#releases[@]} -gt $MAX_RELEASES ]; do
                echo "rm -rf $BASE/releases/${releases[0]}"
                releases=( "${releases[@]:1}" )
            done
            ;;
        ROLLBACK)
            i=$(indice_de "$actual")
            if [ -z "$actual" ] || [ "$i" -le 0 ]; then echo "# No hay una release anterior a la que volver"; continue; fi
            anterior=${releases[$((i - 1))]}
            echo "# Volviendo de $actual a $anterior"
            echo "ln -sf $BASE/releases/$anterior $BASE/current.tmp && mv -T $BASE/current.tmp $BASE/current"
            echo "systemctl reload php8.3-fpm"
            actual=$anterior
            ;;
        ESTADO)
            if [ ${#releases[@]} -eq 0 ]; then echo "# Sin releases"; continue; fi
            echo "# Releases (${#releases[@]}):"
            for r in "${releases[@]}"; do
                [ "$r" = "$actual" ] && echo "#   $r ← current" || echo "#   $r"
            done
            ;;
    esac
done
```

```
*) echo "# Orden desconocida: $orden" ;;  
esac  
done
```

La idea clave del despliegue por releases es separar la preparación (descargar, configurar, instalar dependencias en una carpeta nueva, sin prisa) del cambio de versión, que es solo mover un enlace. Si la preparación falla, la web sigue funcionando con la versión anterior.

El cambio con `ln -sfn` a un enlace temporal y `mv -T` es atómico porque `rename` lo es en Linux: las peticiones que llegan ven la versión vieja o la nueva, nunca una mezcla. Con PHP-FPM se recarga para vaciar la caché de OPcache, que guarda rutas resueltas.

Conservar varias releases es lo que hace instantáneo el rollback, y limitarlas a tres evita llenar el disco. El estado (qué release es la actual) vive en una variable aparte porque, tras volver atrás, ya no es la más reciente.

El modo simulación (escribir las órdenes en lugar de ejecutarlas) es una práctica de seguridad en scripts que tocan producción: se revisa la salida y, si es correcta, se ejecuta con `bash`. Las líneas con `#` son comentarios y no hacen nada al ejecutarla.