qchat — manual de línea de comandos

Para quien instala, ejecuta y maneja un nodo qchat desde una terminal. Cubre los tres programas que produce este repositorio y nada más.

Aplica a	`qchat` 1.0.0
Programas	`qchat` (el nodo), `qchatctl` (su cliente), `qchatsign` (la llave de las liberaciones)
Audiencia	Quien tenga una consola en la máquina que corre el nodo
También como	PDF, adjunto a cada liberación de este repositorio

> **Pendiente.** Este manual describe solo la línea de comandos. La consola web, `qchatui`, es un repositorio aparte y sus pantallas no están documentadas aquí todavía. **Este manual debe extenderse con ellas** cuando esa consola se libere: qué hace cada pantalla, qué muestra que la terminal no, y cómo se espera que una persona pase de una a otra. Hasta entonces, la sección La consola dice solo cómo instalarla y dónde encontrarla.

Contenido

Qué es qchat

qchat es un nodo de la red Cuezero que una persona corre en su propia máquina. Sus conversaciones son cadenas: cada mensaje es una transacción sellada con una clave que solo tienen los miembros de ese momento, cada bloque lo certifican los miembros que escriben, y la historia que un nodo muestra es lo que dice su copia de la cadena. No hay un servidor en medio; las semillas de la red solo presentan unos nodos a otros.

Alrededor de eso, qchat pone lo que un chat necesita y una cadena no tiene: un nombre y un estado, contactos, la diferencia entre un mensaje que salió y uno que fue leído, alguien escribiendo, archivos compartidos en una conversación, archivos guardados en una bóveda que solo abre su dueño, y un explorador que lee las cadenas de la máquina.

Antes de empezar

Requisito	Por qué
Puertos 7420/udp, 7421/udp, 7422/tcp abiertos, o un router que perfore	Sin ellos el nodo no alcanza a nadie. Un router de casa suele poder; uno corporativo puede que no.
Un reloj más o menos en hora	Cada certificado, perfil y token asume que las máquinas coinciden en la hora con minutos de margen.
Minutos de CPU, una vez	Resolver la identidad es caro a propósito al costo de la red de producción. Pasa una sola vez.
Una terminal en la misma máquina, o un túnel SSH	La superficie de control y la consola están en la interfaz local.

Instalación

Desde una liberación, en Debian y derivados:

```
sudo dpkg -i qchat_1.0.0_amd64.deb
```

Eso instala `qchat`, `qchatctl` y `qchatsign` en `/usr/local/bin`, la configuración de ejemplo en `/usr/share/doc/qchat/`, la unidad en `/usr/lib/systemd/system/` y el script de despliegue de la consola en `/usr/share/qchat/`.

Desde una liberación, como AppImage: descárguelo, dele permiso de ejecución, córralo. Lleva el nodo y el cliente y se reemplaza a sí mismo con una actualización verificada.

Desde las fuentes:

```
git clone https://xato.youser.app/xatoxi-capps/qchat.git
cd qchat
./setup_repo.sh --prefix "$PWD/.local/xato" --clean-prefix --clean-deps-build
```

Eso deja los tres programas en `build/`.

Primer arranque, paso a paso

1. La identidad

```
qchat --show-id
```

La primera vez, esto resuelve la identidad, que toma minutos, y la escribe bajo `~/.local/share/qchat/`. Luego imprime dos líneas:

```
node   3f9a…   a lo que se conectan los otros nodos; lo que agrega un contacto
member 8c21…   lo que listan las conversaciones; lo que nombra una invitación
```

Cada llamada posterior imprime las mismas dos líneas de inmediato. **Dé el id de nodo a la gente que deba alcanzarlo, y el id de miembro a quien funde una conversación con usted.**

2. El nodo

```
qchat
```

Sin archivo, se une a la red de producción, sirve su consola en `http://127.0.0.1:7423/` y su superficie de control en `ws://127.0.0.1:7424/ws`, y escribe el token de esa superficie en `~/.local/share/qchat/ws.token` con modo 0600. Corre hasta `Ctrl-C`.

3. Un perfil y un contacto, desde otra terminal

```
qchatctl profile-set "Alice" en el laboratorio
qchatctl friend-invite <id de nodo de Bob> hola, soy Alice
```

Del lado de Bob:

```
qchatctl friend-requests
qchatctl friend-accept <id que imprimió la anterior>
qchatctl friend-list
```

La última muestra a Alice con `ready` cuando el enlace está arriba y `hasProfile` cuando llegó su registro. Solo uno de los dos escribe un identificador: el otro contesta con el id corto de la solicitud.

4. Una conversación

```
qchatctl room-create direct "" <id de miembro de Bob>
qchatctl msg-send <convId> hola Bob
qchatctl msg-history <convId>
```

El archivo de configuración

Ubicación por defecto `/etc/qchat/qchat.cfg`; un nodo que corre una persona suele tener su archivo junto a sus datos y lo nombra: `qchat --config ~/.local/share/qchat/qchat.cfg`. `--config`

nombra otra. Cada ajuste tiene un valor por defecto y un archivo sin ninguno es un archivo válido. El ejemplo en `deploy/qchat.cfg.sample` lista cada ajuste con su valor y un comentario; esta es la forma corta.

Grupo `qchat`	**Por defecto**	**Qué**
`dataDir`	`~/.local/share/qchat`	Raíz de todos los archivos de abajo
`identityPath`, `keyringPath`	`identity.cnis`, `keyring.bin`	La identidad y el par de claves de lectura
`storePath`, `ledgerPath`	`store`, `ledger`	El almacén local y las cadenas
`tokenPath`	`ws.token`	El token de la superficie de control
`runtimeDir`, `uiDocRoot`, `downloadDir`	`run`, `ui`, `downloads`	Archivos derivados, la consola, lo descargado
`storeMapMb`, `ledgerMapMb`	256, 4096	Techos de los dos mapas
`network`	`mainnet`	`mainnet` (`prod` se acepta como grafía antigua), `dev`, `custom` o `standalone`
`bindAddr`, `publicAddr`	`0.0.0.0`, ninguna	Dónde se ata el nodo; qué publica tras un NAT 1:1
`dhtPort`, `relayPort`, `ipv6`	7420, 7422, 1	Los puertos entre pares; 7421 está fijo uno arriba del primero
`maxLinks`	128	Enlaces abiertos a la vez
`puzzleStatic`, `puzzleDynamic`	22, 18	El costo de una identidad; propiedad de la red
`httpPort`, `wsPort`, `wsAddr`	7423, 7424, `127.0.0.1`	La consola, la superficie, dónde se atan
`wsQblock`	1	Servir el explorador en la superficie
`authGraceMs`, `pushMs`, `maxAdminConns`	5000, 1000, 16	Tiempo para presentar el token, periodo de empuje, clientes a la vez
`roundMs`	4000	Una ronda del acuerdo; después de ella propone el siguiente
`fileMaxMb`, `vaultFragLen`	16, 4096	Archivo compartido más grande; tamaño de fragmento de la bóveda
`typingSend`, `typingShow`, `readReceipts`	1, 1, 1	Tres de los interruptores al arrancar
`profilePublic`	0	El cuarto: quién ve su nombre, contactos (0) o todos (1)

Grupo `qchat`	Por defecto	Qué
`cardMs`	600000	Cada cuánto se busca la tarjeta pública de un miembro; la suya sale cada dos
`logLevel`, `logEnabled`	`INFO`, 1	`TRACE`, `DEBUG`, `INFO`, `WARNING` o `ERROR`
`uiVersion`, `uiSha256`, `uiRegistryUrl`	ninguno	Solo para desplegar una consola distinta de la que trae la release; la version lleva su `v`
`updateUrl`	el registro del proyecto	De dónde se lee el manifiesto firmado

Grupo `bootstrap`	Qué
`seed0` … `seed7`	`host:puerto/pin[@asn]`, leído solo cuando `network` es `custom`

Un archivo rechazado nombra el código: `QCHAT_ERR_CFG_PORT`, `QCHAT_ERR_CFG_NETWORK`, `QCHAT_ERR_CFG_LOG`, `QCHAT_ERR_CFG_CENSUS`, etc., y el nodo no arranca.

El nodo: `qchat`

```
qchat [opciones]
  --config <ruta>    Archivo de configuración
  --network <nombre> Unirse a mainnet, dev, custom o standalone
  --standalone       Iniciar una red en vez de unirse a una
  --show-id          Imprimir las identidades de nodo y miembro y parar
  --quiet            Apagar el log; solo respuestas por la salida estándar
  --version          Imprimir la versión y parar
  --help             Imprimir esto y parar
```

`--network` en la línea de comandos gana sobre el archivo. `--standalone` es `--network standalone`. Tanto `SIGINT` como `SIGTERM` lo detienen limpio: las cadenas se cierran, el almacén se vuelca, los puertos se liberan. `SIGHUP` se ignora a propósito; un cambio de configuración es un reinicio.

Lo que hace al arrancar, en orden: lee el archivo, abre la identidad (la resuelve si falta), abre el almacén y las cadenas, construye el nodo, el acuerdo y el ledger, conecta los archivos, carga los interruptores, los perfiles y las conversaciones que seguía, abre los puertos, luego la superficie de control, la consola y el actualizador. Cualquier rechazo en esa cadena lo detiene con el nombre del código en la salida de error.

Los puertos

Puerto	Protocolo	Quién	Para qué
7420	UDP	la red	La tabla de rutas y el saludo
7421	UDP	la red	El plano de datos, fijo uno arriba del primero
7422	TCP	la red	Relevo, cuando no se puede hacer un camino directo
7423	TCP, local	usted	La consola, archivos estáticos
7424	TCP, local	usted	La superficie de control, WebSocket

Los dos últimos **no** deben ser alcanzables desde afuera. La consola es un paquete estático sin estado propio; la superficie es a donde va cada acción, y su única guarda es el token. Administrar desde otra máquina es un túnel SSH:

```
ssh -L 7424:127.0.0.1:7424 -L 7423:127.0.0.1:7423 usuario@maquina
```

con una copia de `ws.token` del lado cercano, nombrada en la configuración del cliente.

Manejarlo: `qchatctl`

```
qchatctl [opciones] <comando> [palabras]
  --config <ruta>   Archivo de configuración (el mismo que lee qchat)
  --json            Imprimir la respuesta como JSON
  --follow          Quedarse suscrito e imprimir cada evento
  --verbose         Log al nivel de información
  --quiet           Apagar el log
  --help            Imprimir esto y parar
```

El cliente lee el mismo archivo de configuración para hallar el puerto y el token, abre la superficie, presenta el token, manda una acción e imprime la respuesta: un campo por línea, o la carga JSON con `--json`. Un rechazo imprime `qchatctl: <nombre del código> (<número>)` en la salida de error y el estado de salida es 1.

`--follow` se suscribe e imprime cada evento según llega, hasta que el nodo cierra el enlace o usted presiona `Ctrl-C`. Con `--json` cada evento es una línea JSON, que es lo que quiere un script.

Cada comando de abajo es una acción del catálogo que sirve la superficie. Las palabras entre corchetes son opcionales; `yes|no` también acepta `1|0`, `true|false`, `on|off`.

Conversaciones

```
rooms
room-create <clase> [nombre] [miembro,miembro...] [rol,rol...]
room-join <convId> <fundador> [token]
room-leave <convId>
room-rename <convId> <nombre...>
room-open <convId> [yes|no]
room-members <convId>
room-invite <convId> [rol] [ttlSec] [maxUses]
member-add <convId> <miembro> [rol]
member-remove <convId> <miembro>
member-role <convId> <miembro> <rol>
```

Clases. `direct` son dos personas y ambas escriben. `group` es un equipo donde cada `writer` escribe y un `reader` lee. `bulletin` es una comunidad donde solo escribe el fundador. La clase se fija al crear.

Roles. `owner` (el fundador, uno por conversación), `writer`, `reader`. Los miembros que escriben certifican los bloques; los lectores tienen la cadena y la clave y no escriben nada.

Fundar. `room-create` escribe el génesis, agrega a cada miembro nombrado, le manda a cada uno un aviso, y rota la clave cuando conoce la clave de lectura de todos. `rooms` muestra `ready` cuando eso terminó; hasta entonces los mensajes se rechazan con `QCHAT_ERR_LGR_KEY_PENDING`. Los miembros nombrados deben ser contactos suyos cuyo perfil usted tiene, y usted debe ser contacto de ellos, o sus nodos descartan el aviso.

Entrar. Un miembro puesto por el fundador entra por el aviso, sin hacer nada. Cualquier otro entra con una invitación: `room-invite` imprime un token (hexadecimal) con un rol, una vida en segundos y un número de usos, y `room-join <convId> <id de miembro del fundador> <token>` del otro lado escribe la entrada en la cadena. El fundador rota entonces la clave para la nueva membresía y le entrega al recién llegado la clave de la época actual. Lo escrito en épocas anteriores queda ilegible para el recién llegado, y `msg-history` dice `no key` en esas entradas. Un token entregado antes de que llegue el primer bloque de la cadena se guarda y se ofrece cuando está, así que la entrada puede pedirse justo después de tener al fundador como contacto.

`room-join` sin token sigue la cadena sin entrar, que es lo que hace un lector de un boletín antes de ser agregado.

El nombre. El nombre de una conversación va en su cadena, sellado con la clave de la época en curso: solo lo leen los miembros, y lo lee cada uno entre por donde entre. Lo escribe el dueño en cuanto la conversación queda lista y otra vez tras cada rotación, de modo que un recién llegado lo ve poco después de recibir su clave. Un reinicio no lo vuelve a escribir si la cadena ya lo tiene para esa época. Solo el dueño lo cambia, con `room-rename <convId> <nombre>`: el nombre nuevo va a la cadena en la pasada siguiente y cada miembro lo toma al aplicarla.

Salir. `room-leave` escribe la salida cuando usted es miembro y deja de seguir en cualquier caso. El fundador no puede salir de una conversación con miembros dentro; quítelos primero, o deje la conversación terminar.

En pantalla. `room-open <convId> yes` dice que el hilo está frente a usted: desde entonces su nodo manda acuses de lectura para él, si el interruptor está encendido, y `no` cuando deja de mirarlo.

Mensajes y sus estados

```
msg-send <convId> <texto...>
msg-history <convId> [desde] [max] [hidden]
msg-delete <convId> <altura> <índice>
msg-recover <convId> <altura> <índice>
typing <convId> [yes|no]
```

`msg-send` sella el texto con la clave de la época actual y se lo entrega al ledger. Responde de inmediato; el mensaje está `sending` hasta que los miembros que escriben certifican su bloque, que en una red tranquila es un segundo o dos.

`msg-history` imprime una página desde el bloque `desde` (1 es el inicio, 0 el valor por defecto) de a lo sumo `max` mensajes (50 por defecto, 200 como máximo), cortada en un límite de bloque para que la página siguiente nunca repita un mensaje; `next` dice dónde empieza la siguiente y `more` si la hay. `hidden yes` incluye los mensajes que usted ocultó. Cada entrada lleva su bloque e índice (lo que toma `msg-delete`), su autor con el nombre con que usted lo conoce, la época, la clase, el texto, y:

`state`	**Qué significa**
`sending`	En el pool; ningún bloque certificado lo tiene todavía
`sent`	En un bloque certificado por los escritores
`delivered`	Todos los demás miembros tienen ese bloque
`read`	Todos los demás miembros lo mostraron en una pantalla

`why`	**Cuando un mensaje no pudo abrirse**
`no key`	Se selló bajo una época de la que usted no tiene clave; quien entró tarde ve esto en lo anterior
`not a member`	Usted no era miembro en esa época
`tampered`	La clave que tiene no abre nada aquí

`msg-delete` oculta un mensaje en esta máquina y en ninguna otra: la cadena es lo que es. `msg-recover` lo vuelve a mostrar.

`typing <convId> yes` les dice a los miembros que usted escribe; el nodo lo repite cada tres segundos mientras dure y los demás dejan de mostrarlo cinco segundos después del último. `no` lo detiene antes.

Personas: contactos, perfiles, presencia

```
friend-invite <nodo> [mensaje...]
friend-remove <nodo>
friend-requests
friend-accept <id>
friend-reject <id>
friend-add <nodo>
friend-list
peer-status <nodo>
profile-get [miembro]
profile-set <nombre> [estado...]
presence-set [typingSend] [typingShow] [readReceipts] [profilePublic]
net-status
```

Un **contacto** se hace con una **solicitud**. `friend-invite <nodo> [mensaje]` le pide a ese nodo ser su contacto: su nodo abre un enlace y, cuando sirve, le manda la solicitud con su perfil firmado y el mensaje (hasta 280 bytes). Del otro lado, `friend-requests` la lista con un `id` corto (los ocho primeros dígitos del id de nodo de quien pide), su nombre, su estado y el mensaje; `friend-accept <id>` lo acepta y le contesta con el perfil propio; `friend-reject <id>` la olvida sin avisar a nadie. Hasta que llegue el sí, la solicitud aparece también en la lista de quien la mandó, con `incoming no`, y se reenvía cada vez que el enlace se recupera. Una solicitud solo se acepta si trae un perfil que verifica contra quien la manda: nadie pide en nombre de otro. Si los dos se piden a la vez, la segunda petición vale como sí. Pedir a quien ya es contacto no manda nada y lo dice: `QCHAT_ERR_PRF_FRIEND`. `friend-remove <nodo>` olvida a un contacto, una solicitud o un conocido solo de este lado y no avisa a nadie: el otro lado lo conserva hasta que también lo olvide, y su siguiente saludo aparece aquí como solicitud de un desconocido. Sirve para repetir una invitación desde cero.

`friend-add <nodo>` es el camino viejo y sigue ahí: hace contacto a ese nodo de este lado solamente y lo saluda con el perfil propio. Quien recibe ese saludo de un desconocido lo ve en `friend-requests` como una solicitud sin mensaje.

Su nodo se conecta a cada contacto y le pide su perfil cada cinco segundos hasta tenerlo. `friend-list` muestra a cada uno con `ready` (el enlace sirve), `hasProfile` y `friend` (ya es contacto, no solo un conocido), y el nombre y el estado cuando se conocen.

Un **perfil** es un registro firmado: nombre, línea de estado, un avatar de hasta 8 KiB, la clave de lectura de ese nodo, y una secuencia. `profile-set` cambia el suyo; cada contacto lo aprende en su próxima petición. `profile-get` sin miembro imprime el suyo; con un id de miembro imprime el de ese, si lo tiene. Un registro cuya firma no verifica, o más viejo que el que se tiene, se descarta.

La **presencia** son cuatro interruptores:

Interruptor	Apagado significa
`typingSend`	Su nodo nunca dice que usted escribe
`typingShow`	Su nodo ignora lo que otros dicen sobre escribir
`readReceipts`	Su nodo no manda marca de lectura, y por eso no recibe ninguna: a quien no manda no se le manda

Interruptor	Apagado significa
`profilePublic`	Solo sus contactos ven su nombre y su estado; los miembros de sus conversaciones ven su id de miembro. Apagado por defecto

`presence-set` sin palabras los imprime; con palabras `yes|no`, en ese orden, los fija, y el cambio se conserva entre reinicios.

Su perfil, con el avatar, va a sus contactos y a nadie más: un miembro con quien solo coincide en una conversación y que se lo pide no lo recibe. Con `profilePublic yes` su nodo publica en la red una **tarjeta** del perfil: nombre y estado, firmados, sin avatar. Cada nodo que comparte una conversación con usted la busca y muestra su nombre. La tarjeta se vuelve a publicar cada veinte minutos y caduca una hora después de la última vez. Con `profilePublic no` su nodo publica en su lugar una tarjeta en blanco, y quien la lee olvida su nombre; un nodo que no llegó a leerla lo olvida cuando caduca la tarjeta pública. Una tarjeta no se puede borrar de los nodos que la guardan, solo sustituir.

`net-status` imprime cómo ve la red a este nodo: `inNetwork`, los enlaces abiertos y usables, los pares que tiene, cómo puede alcanzarlo el exterior (`reach`) y si eso fue confirmado. `peer-status <nodo>` imprime el camino en uso con un contacto.

Archivos y la bóveda

```
file-share <convId> <ruta>
file-list [convId]
file-fetch <fileId>
vault-store <ruta> [nombre] [local|net]
vault-list
vault-fetch <nombre>
vault-export <nombre> <ruta>
vault-import <ruta>
```

Un **archivo compartido** se corta en trozos, se sella con la clave de época de la conversación, se escribe en su cadena como una serie de transacciones, y lo rearma cada miembro con la clave. `file-share` responde con el id del archivo de inmediato; los trozos siguen en los próximos bloques. `file-list` muestra cada archivo con su nombre, tamaño, trozos, cuántos tiene este nodo, y `complete`, `sending` o `failed`. `file-fetch` escribe un archivo completo en el directorio de descargas e imprime la ruta; un nombre ya presente recibe un sufijo numérico. El archivo más grande es `fileMaxMb`.

Un **archivo de la bóveda** se corta en fragmentos sellados con una clave derivada de su identidad. `vault-store <ruta> [nombre]` lo guarda bajo el nombre dado o el propio del archivo; `vault-list` muestra nombres, tamaños, cuentas de fragmentos y dónde están (`where local` o `where network`); `vault-fetch <nombre>` lo vuelve a escribir en el directorio de descargas. Un archivo vacío se rechaza.

Por defecto (`local`) los fragmentos **nunca salen de esta máquina**. Con `net` como tercera palabra se publican además en la caché de registros de los nodos de la red, como registros firmados, y se conserva aquí una copia sellada para poder reofrecerlos: un custodio olvida un registro a la hora si nadie se lo reofrece, y el nodo lo hace cada 20 minutos mientras está

encendido. El trabajo de red lo hace el bombeo, un fragmento por pase y un trabajo a la vez: `vault-store … net` contesta en cuanto acepta el encargo y `vault-list` enseña cómo va (`state storing`, `done`/`total`) y cómo acabó (`state done` con `copies`, el menor número de custodios de un fragmento, o `state failed` con su `code`); la receta solo se guarda si todos los fragmentos encontraron custodio. `vault-fetch` de un archivo de red pregunta primero a la red y después a la copia local: la primera llamada arranca la reconstrucción y contesta `QCHAT_ERR_FIL_PENDING`, y la primera tras acabar entrega la ruta; `served` dice cuántos fragmentos sirvió la red. El tope en red es de 256 KB (`QCHAT_ERR_FIL_NET_SIZE`), con fragmentos de 2048 bytes; otro trabajo en curso da `QCHAT_ERR_FIL_NET_BUSY` y un nodo aún sin tabla de rutas, `QCHAT_ERR_FIL_NET_DOWN`. Cada paso se avisa con el evento `vault-update`.

`vault-export <nombre> <ruta>` escribe la **receta sellada** de un archivo y `vault-import <ruta>` la recoge. La receta sale como está guardada: solo la abre la identidad que la selló, así que es segura de llevar e inútil para cualquier otro. Es lo que necesita otra máquina con la misma identidad para recuperar un archivo cuyos fragmentos están en la red.

El explorador

```
chain-list
chain-info <convId>
block-list <convId> [desde] [max] [desc]
block-show <convId> <altura>
tx-list <convId> <altura>
tx-show <convId> <altura> [índice]
chain-verify <convId> [altura] [índice]
```

El explorador lee las cadenas de esta máquina, decodifica lo que puede y dice lo que no. Se sirve solo cuando la superficie se ata a la interfaz local y `wsQblock` es 1; si no, cada uno de estos responde `QCHAT_ERR_BLK_OFF`.

- `chain-list`: cada cadena que se tiene, con su nombre, clase, punta, bloques aplicados y bytes.
- `chain-info`: una cadena en detalle: fundador, época, miembros, validadores, hashes de punta y génesis, particiones.
- `block-list`: una página de bloques desde `desde` (1 es el génesis), a lo sumo `max` (100 por defecto, 500 como máximo), ascendente salvo `desc`; cada fila con su hash, hora, transacciones y `hasCert`, si se guarda un certificado para él. Una página no comprueba ninguna firma.
- `block-show`: un bloque entero: la fila, sus comprobaciones de encadenado y Merkle y su certificado juzgado como lo juzga el ledger: `writers`, quiénes podían escribir a la altura de debajo del bloque; `quorum`, cuántos de ellos tenían que firmar; `certified`, si esa cantidad de firmas se sostiene; y cada firmante con `verified`, si su firma se sostiene. Nada de eso depende de quién es miembro hoy. Para contestar se recorre la cadena hasta el bloque, así que el costo crece con la altura.
- `tx-list` y `tx-show`: las transacciones de un bloque y una decodificada: una fundación, un miembro agregado o quitado, una clave rotada, un mensaje abierto cuando usted tiene la clave y marcado `hidden` si lo ocultó.
- `chain-verify`: valida la cadena desde el génesis, audita cada certificado, y con una altura y un índice prueba esa transacción contra la raíz Merkle de su bloque.

La identidad

```
identity-export <path> <passphrase>
identity-import <ruta> <frase>
```

`identity-export` escribe la identidad y el par de claves de lectura en un archivo sellado con una frase que pide (ocho caracteres al menos) e imprime la ruta. `identity-import` abre ese archivo en otra instalación y escribe la identidad junto a la de ese nodo; los archivos que ya estaban se conservan con sufijo `.bak`. El nodo en marcha conserva la identidad con la que arrancó; la importada se usa desde el próximo arranque. La frase equivocada es `QCHAT_ERR_IDN_PASSPHRASE`; un archivo que no es una exportación es `QCHAT_ERR_IDN_FORMAT`.

Dos máquinas corriendo la misma identidad a la vez confunden a todos los que la tienen como contacto. Importe, luego detenga la vieja.

Actualizaciones

```
update-check
update-apply [appimage|deb]
config-show
```

El nodo lee un manifiesto firmado de `updateUrl` treinta segundos después de arrancar y cada seis horas, verifica la firma contra la llave compilada en él, y empuja `update-available` a cada suscriptor cuando la versión es más nueva. `update-check` lo hace ahora e imprime el manifiesto con `verified` y `newer`. `update-apply` descarga el artefacto de la clase pedida (o la que corresponde a cómo corre el nodo), comprueba su digesto y tamaño, y: para un AppImage, reemplaza el archivo desde el que corre el nodo y pide un reinicio; para un `.deb`, lo deja en el directorio de descargas e imprime la ruta para `dpkg -i`.

Una compilación cuya llave es la de relleno rechaza todo manifiesto con `QCHAT_ERR_UPD_UNSIGNED`; un manifiesto cuya firma no verifica es `QCHAT_ERR_UPD_SIG`; un artefacto con digesto equivocado es `QCHAT_ERR_UPD_HASH` y se borra. Un manifiesto verificado puede traer un censo nuevo de semillas; se escribe en `downloads/seeds.cfg` y se lee en el próximo arranque.

`config-show` imprime los ajustes efectivos, sin el token.

La consola

La consola es un paquete estático servido en `http://127.0.0.1:7423/`. No está en este repositorio, pero **viaja dentro de la release**: una instalación recién hecha sirve la consola que vino con el programa, sin configurar nada y sin red, y actualizar el programa actualiza la consola con él, porque las dos van en el mismo artefacto firmado. No hay nada que hacer.

El nodo elige en este orden y sirve la primera que encuentra:

1. lo que haya desplegado en `uiDocRoot`, porque quien lo puso lo quiso así;
2. la consola que trae la instalación (`/usr/share/qchat/ui`, o dentro de la imagen portable);
3. la página de relleno, que dice qué está pasando.

Desplegar una consola distinta de la que trae la release es el camino del operador, y ahí sí hacen falta `uiVersion` y `uiSha256`:

```
./deploy/fetch-ui.sh --config ~/.local/share/qchat/qchat.cfg
```

La versión es la **etiqueta de publicación, con su `v`**: escrita sin ella el registro contesta 404, que se lee como una caída y es una errata. Un digesto que no coincide no despliega nada. La consola habla con la superficie con el token; hace lo mismo que `qchatctl`, porque hay un solo catálogo.

Recetas de todos los días

¿Está vivo?

```
qchatctl status
qchatctl net-status
```

Mirar todo lo que pasa:

```
qchatctl --follow --json subscribe
```

Fundar un equipo, invitar a alguien, verlo entrar:

```
qchatctl room-create group "equipo"
qchatctl room-invite <convId> writer 86400 5
# dele el token a Carol; ella corre:
qchatctl room-join <convId> <su id de miembro> <token>
# usted corre:
qchatctl room-members <convId>
```

Leer los veinte bloques más nuevos de una cadena:

```
qchatctl block-list <convId> 0 20 desc
```

Mudarse a otra máquina:

```
qchatctl identity-export ./yo.qid "la frase"
# copie el archivo; en la máquina nueva:
qchatctl identity-import ./yo.qid "la frase"
# detenga el nodo viejo, arranque el nuevo
```

Correrlo como servicio en una máquina sin nadie delante: vea `deploy/qchat.service`, que explica el directorio de estado, el usuario y el reloj.

Cuando algo anda mal

Síntoma	Mire
`qchatctl` dice `QCHAT_ERR_CTL_CONNECT`	El nodo no corre, o el `wsPort` del archivo del cliente no es el del nodo
`qchatctl` dice `QCHAT_ERR_CTL_DENIED`	El archivo de token que lee el cliente no es el del nodo
`net-status` dice `inNetwork 0` por minutos	Los puertos entre pares están bloqueados, o el censo está mal; el log nombra las semillas que probó
Un contacto nunca llega a `ready`	Está apagado, o ninguno de los dos es alcanzable y el relevo se rechaza; `peer-status` dice el camino
Un contacto nunca llega a `hasProfile`	Su nodo no lo agregó a usted; un perfil se responde a contactos
`rooms` nunca dice `ready`	La clave de lectura de un miembro es desconocida: no es un contacto con perfil
`msg-send` dice `QCHAT_ERR_LGR_KEY_PENDING`	La rotación no se aplicó todavía; espere `ready`
Un mensaje queda `sending` dos minutos y desaparece	Ningún quórum lo certificó: los escritores están apagados. Se descarta del pool; mándelo de nuevo
`msg-history` dice `no key`	Esa época fue antes de que usted entrara; nada que arreglar
`chain-list` dice `QCHAT_ERR_BLK_OFF`	La superficie no está en la interfaz local, o `wsQblock` es 0
`update-check` dice `QCHAT_ERR_UPD_UNSIGNED`	Esta compilación no lleva llave de liberación; vea `doc/release-signing.md`

El log está en la salida de error de `qchat`; `logLevel = "DEBUG"` dice por qué pasó cada rechazo.

Códigos de error

Cada respuesta rechazada lleva el nombre y el número de un código.

Familia	Rango	Significado
`QCHAT_ERR_GEN_*`	-1 … -10	Un nulo, un valor inválido, un rango, un búfer chico, no encontrado, estado equivocado, tiempo agotado, lleno, existe, denegado
`QCHAT_ERR_MEM_*`	-101	No se pudo reservar memoria
`QCHAT_ERR_CFG_*`	-201 … -210	La configuración: archivo, ruta, puerto, dirección, límite, censo, consola, log, red, almacén

Familia	Rango	Significado
`QCHAT_ERR_IDN_*`	-301 … -311	La identidad: falta, inusable, no resuelta, no guardada, existe, cancelada, exportación, importación, frase, formato, llavero
`QCHAT_ERR_STO_*`	-401 … -405	El almacén local: abrir, escribir, leer, formato, lleno
`QCHAT_ERR_NET_*`	-501 … -504	El nodo: rechazó, un envío falló, no listo, no existe el par
`QCHAT_ERR_LGR_*`	-601 … -611	El ledger: crear, conversación, entrada, historia, clave, clave pendiente, bifurcada, miembro, rol, invitación, firmantes
`QCHAT_ERR_ROOM_*`	-701 … -709	Una conversación: clase, nombre, no es miembro, llena, cerrada, existe, id malformado, no encontrada, solo lectura
`QCHAT_ERR_MSG_*`	-801 … -808	Un mensaje: formato, descifrado, sin clave, no es miembro, muy grande, oculto, no es de chat, no encontrado
`QCHAT_ERR_KEY_*`	-901 … -905	Las claves de lectura: envolver, desenvolver, sin clave del par, formato, falló una primitiva
`QCHAT_ERR_PRF_*`	-1001 … -1005	Un perfil: formato, firma, tamaño, viejo, desconocido
`QCHAT_ERR_PRS_*`	-1101 … -1103	Presencia: formato, clase, el interruptor está apagado
`QCHAT_ERR_BLK_*`	-1201 … -1206	El explorador: cadena, altura, transacción, prueba, apagado, página muy grande
`QCHAT_ERR_FIL_*`	-1301 … -1309	Archivos: drop, bóveda, ruta, nombre, tamaño, pendiente, rotado, no encontrado, incompleto
`QCHAT_ERR_WS_*`	-1401 … -1409	La superficie: token, sin autenticar, esquema, acción, argumento, conexiones, remota, envío, JSON
`QCHAT_ERR_CTL_*`	-1501 … -1504	El cliente: conexión, respuesta, denegado, uso
`QCHAT_ERR_UPD_*`	-1601 … -1609	El actualizador: descarga, manifiesto, firma, hash, versión, aplicar, formato, compilación sin llave, ya al día

Familia	Rango	Significado
`QCHAT_ERR_WEB_*`	-1701 ... -1702	La consola: raíz de documentos, arranque
`QCHAT_ERR_SUB_*`	-1801 ... -1809	Un componente de abajo rechazó
`QCHAT_ERR_SYS_*`	-1901 ... -1906	El sistema: mutex, reloj, archivo, señal, hilo, socket

---

Límites conocidos

- Un perfil se aprende de un contacto y de nadie más; no hay un directorio donde buscar a una persona.
- La bóveda guarda sus fragmentos solo en esta máquina.
- Solo el fundador rota la clave de una conversación, y a quien entra tarde se le entrega la época actual y ninguna anterior.
- El explorador se lee en la máquina donde vive la cadena.
- Esta compilación lleva la llave de actualización de relleno hasta que la llave de liberación se fije en ella.
- La consola web se documenta en su propio repositorio cuando se libere.