Introduction to OpenSSH

Darren Tucker <dtucker@openssh.com>

What is OpenSSH?

A suite of programs providing encrypted remote login, file transfer and tunnelling.

Provides

- **confidentiality** - no one can **observe** what's sent
- **integrity** - no one can **change** what's sent (without being detected)
- **identity** - both ends **are** who they claim to be

An implementation of the Secure Shell (aka SecSH or SSH) protocol and related tools: RFC4250 - 4256, and others. See http://openssh.com/specs.html

This describes v2 of the SSH protocol: v1 is obsolete and largely no longer used.

Remote login: `ssh` and `sshd`

Simplest use case: logging into a remote server. Your `ssh` connects to `sshd` running as a daemon on the server, which forks a copy of itself* to handle you.

```
client$ ssh server
dtucker@server's password:
Last login: Mon Aug  4 21:00:57 2025 from 192.168.1.1
server$
```

This allocates a *pseudoterminal (pty)* which enables advanced terminal IO and shell features like job control.

… but we will be ignoring ptys for now.

* In recent versions, sshd is actually a set of independent cooperating binaries. We will be ignoring that for now too..

Remote login: `ssh` and `sshd`

```
client$ ssh server
The authenticity of host 'server (192.168.1.1)' can't be established.
ED25519 key fingerprint is SHA256:FDsILxe[...].
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

A server's **host key** proves its identity. The **host key fingerprint** is a hash of the private key and uniquely identifies a key.

Fingerprint is learned on first connection, or supplied out of band, stored in `~/.ssh/known_hosts` and verified each subsequent connection.

By default OpenSSH uses the Trust On First Use (TOFU) or “duckling” model.

Remote login: host keys

Early in each connection, `ssh` checks the host's key matches the expected value in `~/.ssh/known_hosts` or `/etc/ssh/ssh_known_hosts`.

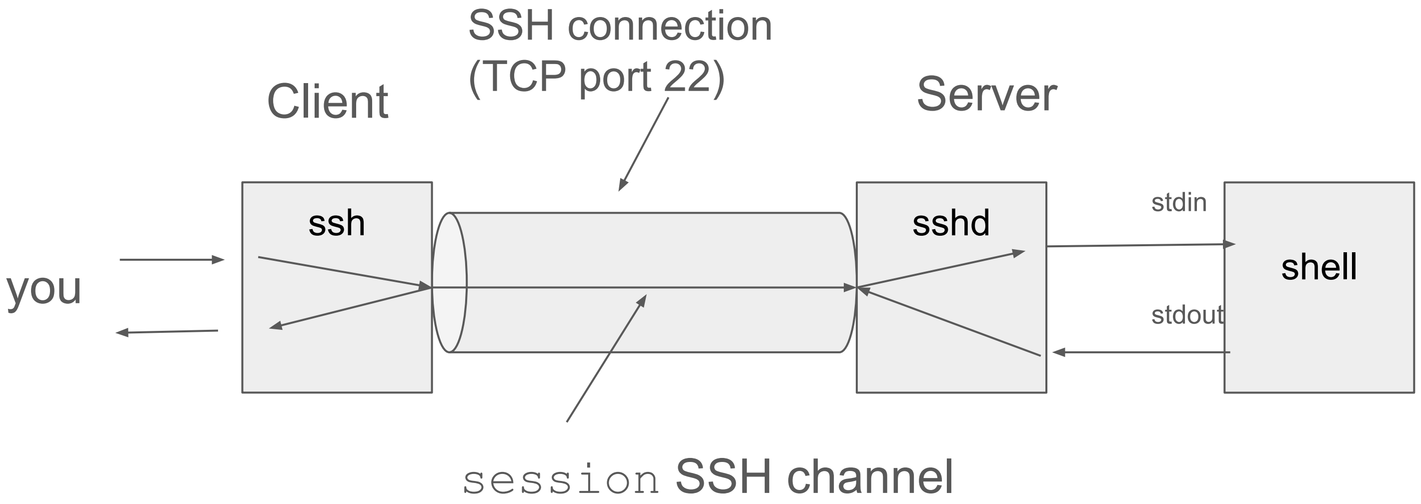
```
client$ ssh server
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
Host key for server has changed and you have requested strict checking.
Host key verification failed.
```

Host key verification is the only thing preventing someone else pretending to be your server!

Remote login: under the covers

Your request creates a session channel inside the encrypted SSH connection. A single SSH connection may have many channels of various different types.

```
client$ ssh server
Last login: Mon Aug  4 21:00:57 2025 from 192.168.1.1
server$
```



Remote login: escape character

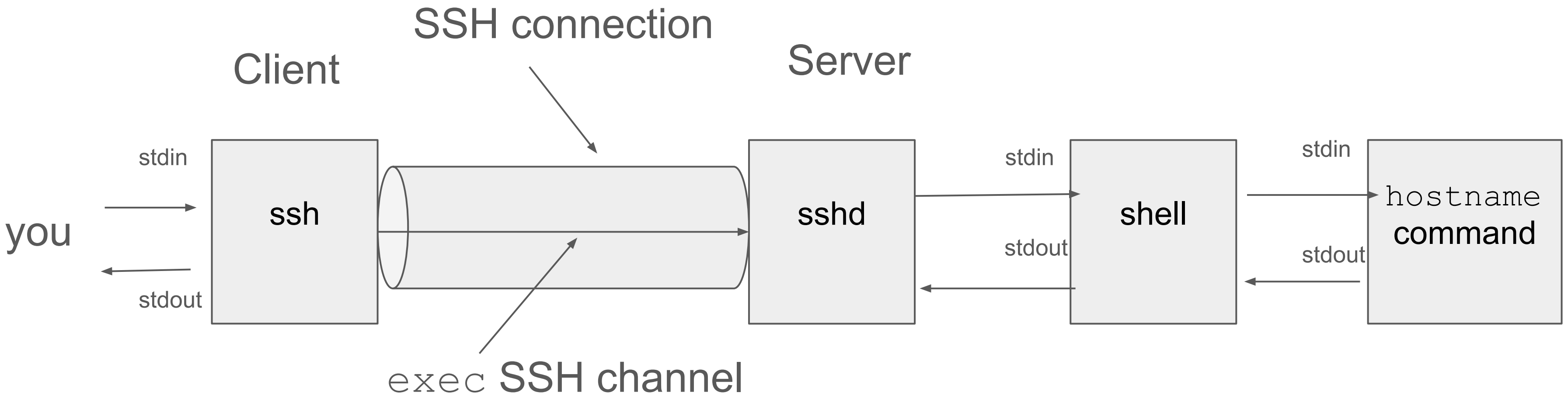
- Normally anything you send to ssh gets passed straight through to the other end.
- sometimes you want to interact with your local ssh
  - eg to close it in case of network problems.
- ssh has an `EscapeChar`, by default "**~**" but can be changed or disabled
- only recognised after a newline

```
server$ ~?
Supported escape sequences:
 ~.   - terminate session
 ~B   - send a BREAK to the remote system
 ~R   - request rekey
 ~#   - list forwarded connections
 ~?   - this message
 ~~   - send the escape character by typing it twice
(Note that escapes are only recognized immediately after newline.)
```

Remote commands

If given a command, ssh runs it on the server, passes stdin to it and returns its stdout to the client.

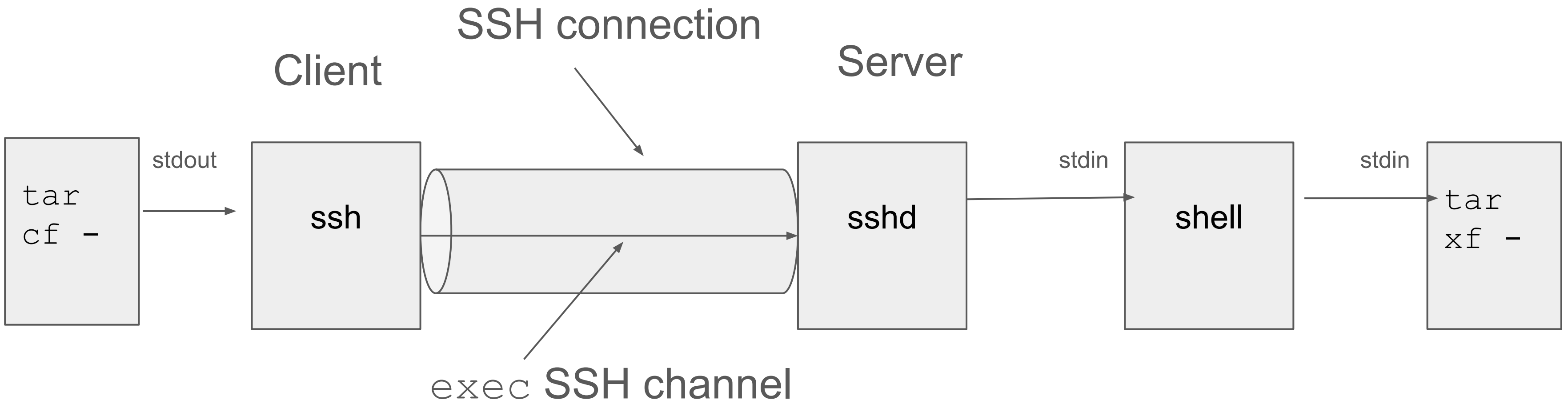
```
client$ ssh server hostname
server.dtucker.net
client$
```



Remote commands: pipelines

You can use ssh to make pipelines span systems, eg

```
$ tar cf - somedir | ssh server '(cd dir2 && tar xf -)'
```



Remote commands: arguments and quoting

If not quoted, the "command" will be concatenated, separated by single spaces:

```
dtucker@client$ ssh server echo Hello,               world.
Hello, world.
```

If quoted, it can be a complex shell command, including pipelines:

```
dtucker@client$ ssh server 'for i in *; do file $i; done'
```

Note that quoting can change local vs remote shell expansion:

```
dtucker@client$ ssh root@gate "echo $LOGNAME"
dtucker
```

```
dtucker@client$ ssh root@gate 'echo $LOGNAME'
root
```

File transfer: `sftp` and `sftp-server`

`sftp` copies files interactively or as specified on the command line using `ssh` as the transport.

```
client$ sftp server
Connected to server.
sftp> cd /tmp
sftp> put local-file /tmp/remote-file
Uploading local-file to /tmp/remote-file
local-file                                   100% 1139   124.9KB/s   00:00
```

Other clients also use `sftp-server`, eg `sshfs` which presents a remote sftp server as a local filesystem.

File transfer: `scp`

`scp` copies files using ssh as a transport, supporting any combination of {local,remote} to {local,remote}

```
client$ scp local-file server:remote-file
client$ scp server:remote-file local-file
client$ scp server1:remote-file server2:remote-file
```

By default, remote-to-remote transferred via the client.

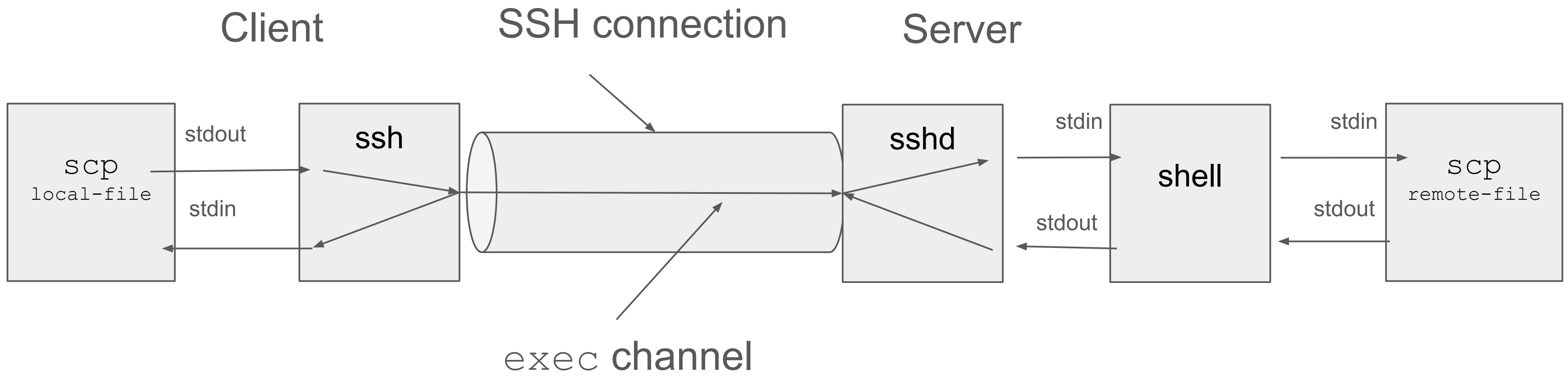
Originally the same protocol as `rcp`, now actually SFTP under the covers.

Old protocol available via `scp -O`.

SSH as a transport layer

In addition to pipelines, many tools use `ssh` as an authenticated network transport, eg scp, sftp, git, cvs, rsync.

```
client$ scp -O local-file server:remote-file
```



SSH Authentication

SSH supports many authentication methods, including:

- `password`
- `keyboard-interactive`
- `public-key`
    - `plain keys`
    - `certificates`

You can specify specific methods on the the client:

```
$ ssh -oPreferredAuthentications=password server
```

A warning for Internet-accessible systems

An internet-facing sshd will see many, many login attempts from password guessing miscreants. In 2 months mine has seen 26k login attempts across 4.5k distinct users from 16k distinct IP addresses.

On such a system:

- strongly consider disabling `PasswordAuthentication` and `KbdInteractiveAuthentication` and using only keys, otherwise
- **all** accounts must have strong passwords.
- recent OpenSSH versions have `PerSourcePenalties` which can slow these down somewhat
- third party solutions such as `fail2ban` also exist.

Public key authentication

To set up, you use `ssh-keygen`:

- it creates a key in two parts: private and public
- you install public key on server(s), keep private key.

To use (`ssh` does this for you):

- client asks server if it would accept key
- if so, client requests a "challenge" from server
- client signs challenge with private key, sends back
- server verifies signature, if good auth is allowed

Public key authentication: create key pair

```
$ ssh-keygen
Generating public/private ed25519 key pair.
Enter file in which to save the key (~/.ssh/id_ed25519):
Enter passphrase for "~/.ssh/id_ed25519" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ~/.ssh/id_ed25519
Your public key has been saved in ~/.ssh/id_ed25519.pub
[...]
```

```
$ ls -l ~/.ssh/id*
-rw-------  1 dtucker  wheel  419 Nov 24 13:53 /home/dtucker/.ssh/id_ed25519
-rw-r--r--  1 dtucker  wheel  105 Nov 24 13:53 /home/dtucker/.ssh/id_ed25519.pub
```

Public key authentication: install public key

Add the client's public key to the `~/.ssh/authorized_keys` file on the server:

```
client$ cat /home/dtucker/.ssh/id_ed25519.pub
ssh-ed25519 AAAAC3NzaC1[...]EbqKaLnNl9iGv8See dtucker@server

client$  ssh server <.ssh/id_ed25519.pub 'cat >>.ssh/authorized_keys'
dtucker@server's password:

client$ ssh server
Enter passphrase for key '/home/dtucker/.ssh/id_ed25519':
Last login: Mon Aug  4 21:00:57 2025 from 192.168.1.10
```

`ssh-copy-id(1)` can automate this

Public key authentication: ssh-agent

`ssh-agent` keeps a decrypted copy of your key and will sign challenges on your behalf. `ssh` contacts it via `$SSH_AUTH_SOCK`.

```
client$ eval `ssh-agent`; echo $SSH_AUTH_SOCK
Agent pid 30100

/tmp/ssh-qAHbQSziOgh8/agent.20026

client$ ssh-add
Enter passphrase for /home/dtucker/.ssh/id_ed25519:
Identity added: /home/dtucker/.ssh/id_ed25519 (dtucker@server)

client$ ssh server
Last login: Mon Aug  4 21:00:57 2025 from 192.168.1.10
server$
```

Your desktop environment might start ssh-agent or compatible agent for you, so check if `$SSH_AUTH_SOCK` exists before starting another `ssh-agent`.

Public key authentication: agent forwarding

- Agent forwarding allows the ssh command on remote servers to pass challenges back to your `ssh-agent`.
- This allows it to perform public key authentication without exposing copies of your private key.
  - however a hostile server can **use** your private key via an agent forward!

```
client$ ssh -oForwardAgent=yes server1
Last login: Mon Aug  4 21:00:57 2025 from 192.168.1.10
server1$ ssh server2
Last login: Mon Aug  4 22:03:16 2025 from 192.168.1.11
server2$
```

Configuration files

The `ssh` client has configuration files in 2 locations:

- system-wide (eg `/etc/ssh/ssh_config`)
- per-user `(eg ~/.ssh/config)`
- some other files in both places, eg keys in `~/.ssh/`

The `sshd` server has 1 system-wide config file at `/etc/ssh/sshd_config` plus other files such as host keys at `/etc/ssh/ssh_host_*_key` (and corresponding `.pub`). Server restart needed to pick up changes.

These are described in the `ssh_config(5)` and `sshd_config(5)` man pages.

Config file parsing: first-match per keyword

Wildcard first: override

```
Host *
    Username fred

Host server1
    Username barney
```

Wildcard last: default

```
Host server1
    Username fred

Host *
    Username barney
```

Which username is used for:

- `ssh server1` ?
- `ssh server2` ?

Resources

- this deck at available at https://dtucker.net/openssh/
- man pages, in particular:
    - ssh(1) sshd(8), scp(1), ssh_config(5) sshd_config(5).
- main web site: https://www.openssh.com or https://www.openssh.org
    - mailing lists: https://www.openssh.com/list.html
- bug reports:
    - for vendor-provided binaries: to your OS vendor first
    - https://www.openssh.com/report.html

Advanced topics: debugging `ssh`

Example: our public-key auth isn't working:

```
$ ssh localhost
dtucker@localhost.dtucker.net's password:
```

Both ssh and sshd have a LogLevel in their configs. ssh can also have its debug level raised with the -v option, and sshd with -d, and more -v or -d equals more debugging (up to 3).   Note that both can be quite chatty.

Advanced topics: debugging `ssh`

```
$ ssh -v server
debug1: Will attempt key: /home/dtucker/.ssh/id_ed25519
ED25519 SHA256:futYVCudg4xI1K2Quo4hIN0DSCxgAoKSatA6Gb2TgM0
explicit agent
debug1: Offering public key: /home/dtucker/.ssh/id_ed25519
ED25519 SHA256:futYVCudg4xI1K2Quo4hIN0DSCxgAoKSatA6Gb2TgM0
explicit agent
debug1: Next authentication method: password
dtucker@localhost.dtucker.net's password:

debug1: Remote: Ignored authorized keys: bad ownership or
modes for directory /home/dtucker/.ssh
server$
```

Advanced topics: debugging `sshd`

For more detail you can run the server in a one-shot debug mode, typically on a non-default port, then point a client (**`ssh -p 2222 server`**) at that port.

```
$ sudo /usr/sbin/sshd -d -p 2222
[...]
debug1: trying public key file
/home/dtucker/.ssh/authorized_keys
debug1: fd 7 clearing O_NONBLOCK
Authentication refused: bad ownership or modes for directory
/home/dtucker/.ssh
[...]
Failed publickey for dtucker from 127.0.0.1 port 34156 ssh2:
ED25519 SHA256:[...]
```

Advanced topics: forwarding

In addition to connecting stdin and stdout across machines, OpenSSH has a number of other types of forwarding.

Standard:

- Local and remote TCP forwarding
- X11 forwarding

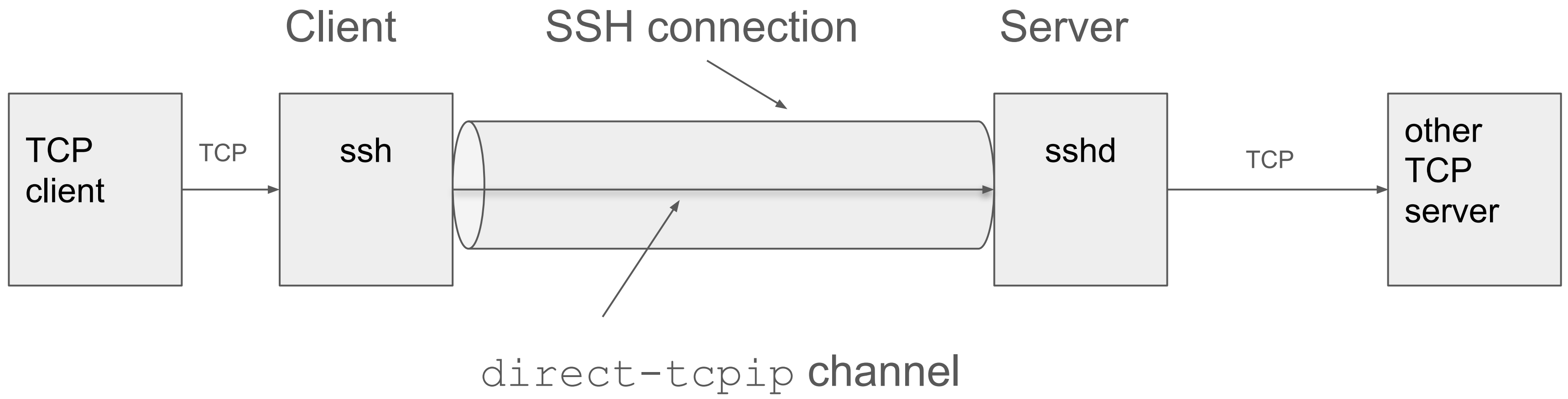
Vendor extensions:

- Unix domain socket forwarding
- ssh-agent forwarding
- IP tunnelling

Port forwarding: local forward

Local port forwarding specifies destination at ssh connection time[*].

- client accepts a TCP connection
- client creates `direct-tcpip` “channel” inside SSH connection
- server connects to specified server at the other end
- everybody passes bytes



Port forwarding: local forward in action

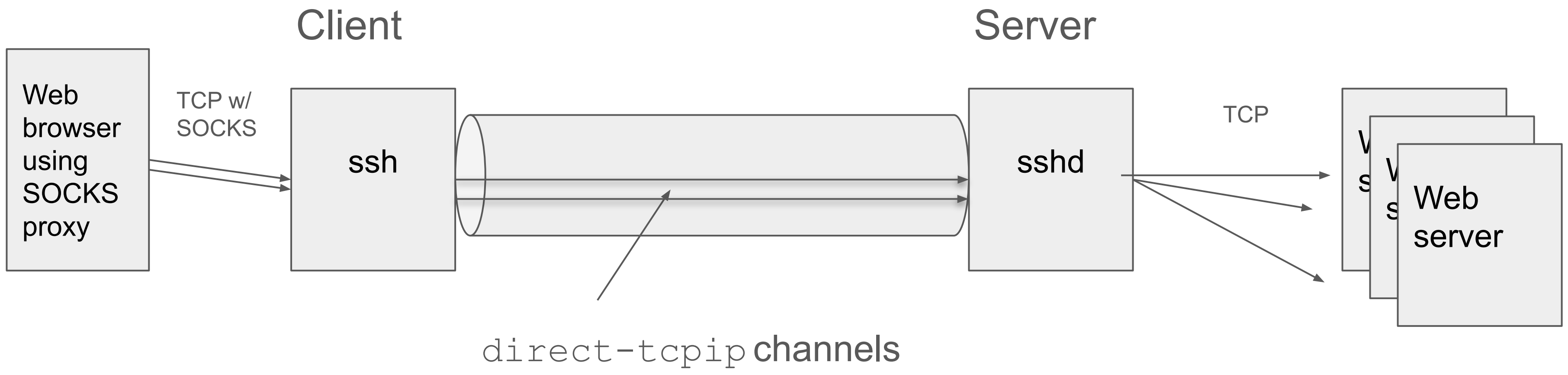
```
client$  ssh -L 1234:127.0.0.1:22 server
[...]
server$

client-shell2$ $  lsof -n -i :1234
COMMAND     PID    USER   FD   TYPE   DEVICE SIZE/OFF NODE NAME
ssh     1621661 dtucker    4u  IPv6 13173014      0t0  TCP [::1]:1234 (LISTEN)
ssh     1621661 dtucker    5u  IPv4 13173015      0t0  TCP 127.0.0.1:1234 (LISTEN)
```

[todo: add server-side here]

Port forwarding: dynamic forward

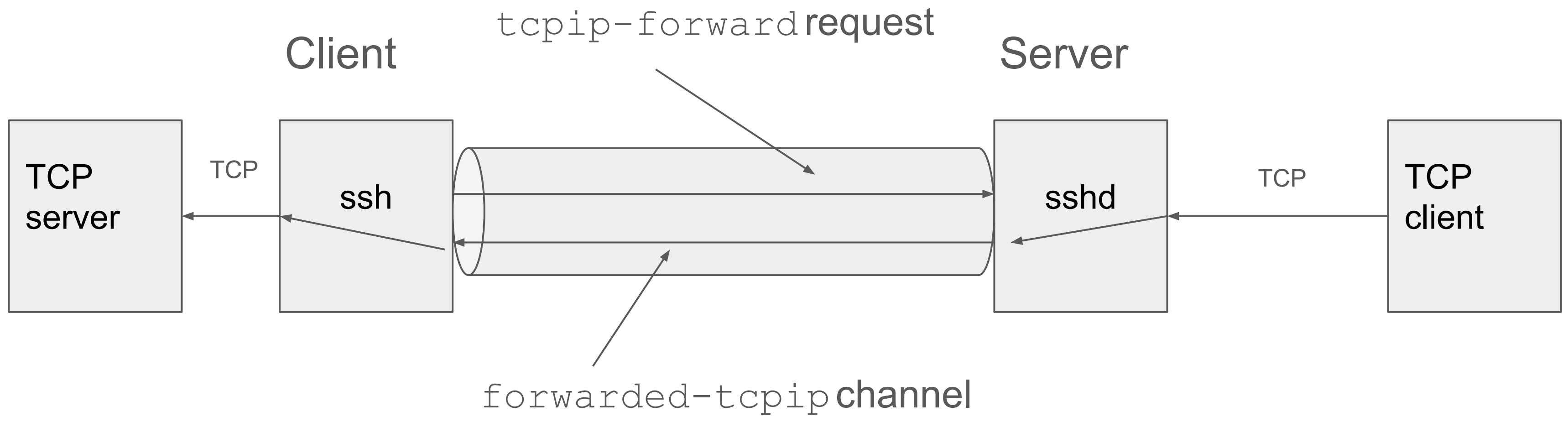
Dynamic forwarding allows client to specify destination via SOCKS (which most browsers support) in each request header. For each request:

- client accepts a TCP connection on client, reads destination
- client creates SSH connection “channel” requesting specified host:port
- server connects to requested server at the other end
- everybody passes bytes



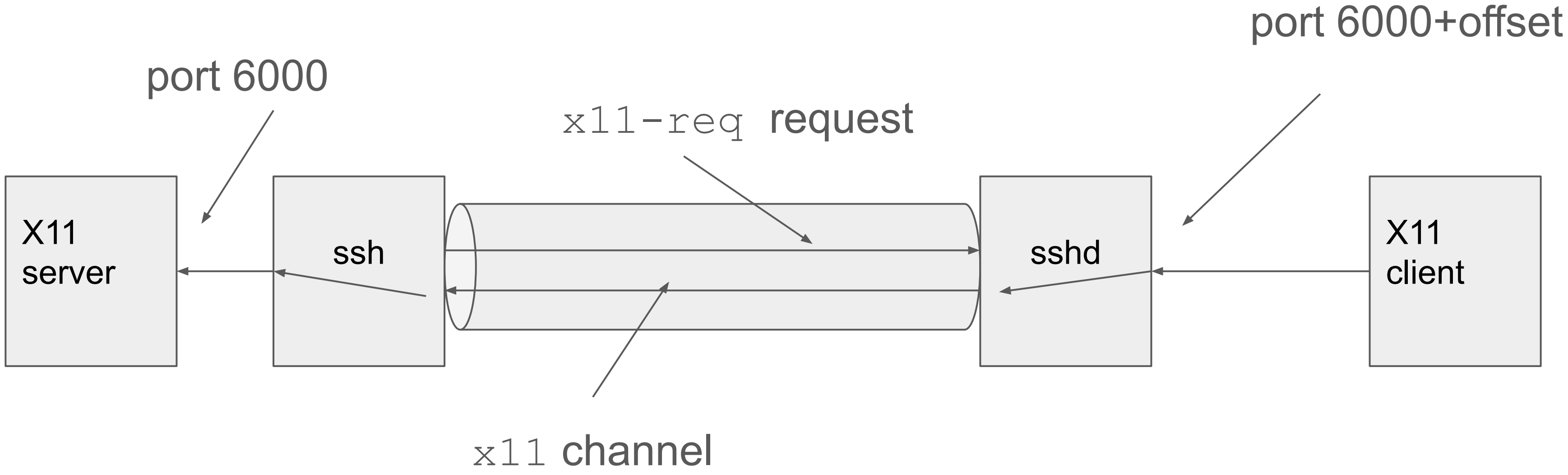
Port forwarding: remote forward

- client requests server listen on specific port by sending `tcpip-forward`
- for each TCP connection received on port
  - server creates `forwarded-tcpip` channel
  - client makes the specified connection at the other end
  - everybody passes bytes



Port forwarding: X11 forward

- a special case of remote forwarding
- adds xauth authentication
  - but note that enabling X11 forwarding on an untrustworthy server is extremely risky



Port forwarding: X11 forward

```
client$ echo $DISPLAY
localhost:0
client$ ssh -oForwardX11=yes server
/usr/X11R6/bin/xauth:  file ~/.Xauthority does not exist

server$ xauth list
server/unix:10  MIT-MAGIC-COOKIE-1  ffbf41[..]f342eb18
server$ echo $DISPLAY
localhost:10.0
server$ xeyes
```

Other advanced features

- `ProxyCommand / LocalCommand`
- connection multiplexing
- jumphosts
- config file Match
- FIDO2 (hardware) keys
- TOKEN expansion
- key restrictions

Questions?

- This was only a brief overview. There are many other features and options.

- man pages, in particular:
    - ssh(1) sshd(8), ssh_config(5) sshd_config(5).
- main web site: https://www.openssh.com or https://www.openssh.org
    - mailing lists: https://www.openssh.com/list.html
    - release notes: https://www.openssh.com/releasenotes.html
- bug reports:
    - for vendor-provided binaries: to your OS vendor first
    - otherwise https://www.openssh.com/report.html

Thanks to Marien Zwart and Damien Miller for reviewing this presentation.

Backup slides only beyond this point.

Port forwarding: Unix domain sockets

- Unix domain socket forwarding is not part of the standard
  - (a "vendor extension")
- Works the same way as Local or Remote forwarding
  - except it listens on a Unix domain socket (which exist on a filesystem) instead of TCP.

Port forwarding: Agent forwarding

Agent forwarding is a special case of Unix domain socket forwarding.

```
client$ eval `ssh-agent`
Agent pid 8658

client$ echo $SSH_AUTH_SOCK
/home/dtucker/.ssh/agent/s.QG7fzTtldO.agent.PHYVROiEiN

client$ ssh -A server
[...]

server$ echo $SSH_AUTH_SOCK
/home/dtucker/.ssh/agent/s.XIOSleZIbu.sshd.JPllBAN8Il

server$ sudo lsof $SSH_AUTH_SOCK
COMMAND      PID    USER   FD   TYPE          DEVICE SIZE/OFF     NODE
NAME
sshd-sess 220034 dtucker    8u  unix 0x000000008f8006f7      0t0 28507208
/home/dtucker/.ssh/agent/s.XIOSleZIbu.sshd.JPllBAN8Il type=STREAM (LISTEN)
```