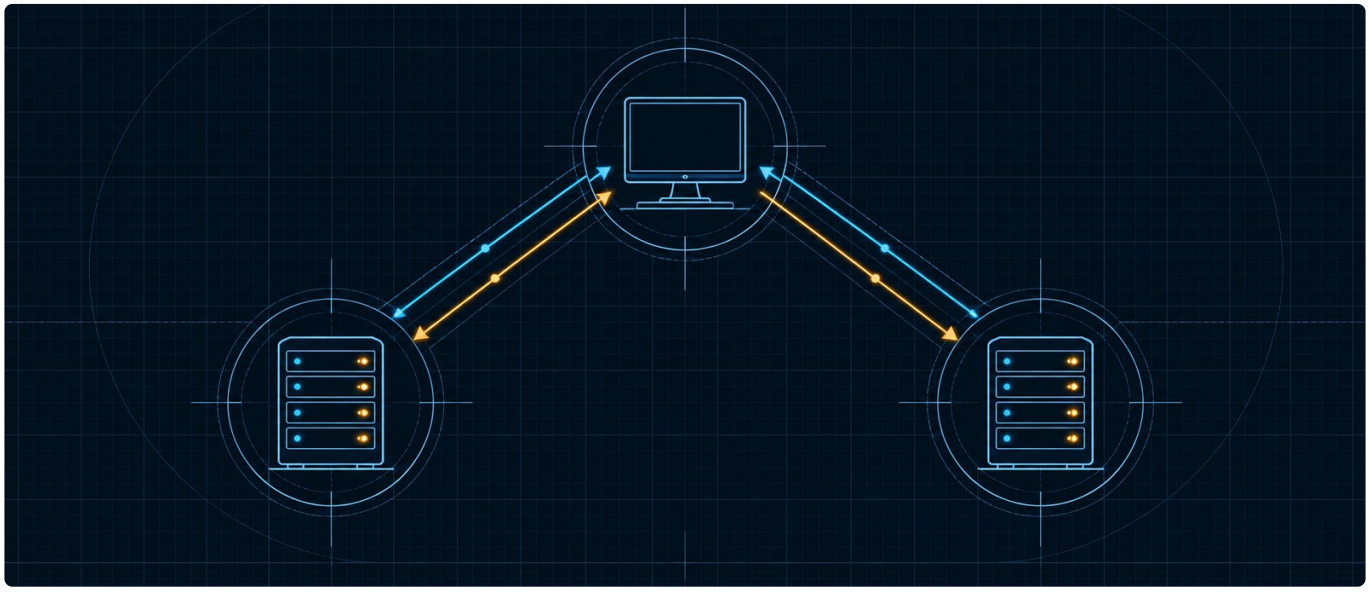
I Have Used scp for 30 Years and Never Knew About -3

October 28, 2026 / Gavin Jackson

ssh scp linux security sysadmin



The -3 flag in one picture: the bytes go through your workstation.

I have been copying files with `scp` since the mid nineties. It is one of those commands I could type in my sleep: source, destination, done. So I was genuinely surprised last week when I found out about the `-3` option, which copies a file from one remote host to another remote host, passing the data through the machine you are sitting at. Thirty years of daily use and the flag had been sitting in the man page the whole time. It was added in OpenSSH 5.7 in January 2011, so it has only been fifteen years of ignorance rather than thirty. Small comfort.

That discovery sent me back to the man page to see what else I had missed. This post is what I found: how scp actually works, the flags worth knowing, how ssh-agent ties it all together, and why the features that make ssh so useful on your own machines are the same ones you should be switching off on production servers.

So what is scp, really?

scp shipped with the original SSH in 1995 as a secure replacement for `rcp`, the Berkeley remote copy command. The important part is that scp was never really its own protocol. For most of its life, when you ran `scp file host:/path/`, your local scp simply ssh'ed to the far end, started a hidden scp process there in source or sink mode, and the two scp processes talked the old rcp wire protocol to each other inside the encrypted ssh channel. All of the hard parts, authentication, encryption, connection

handling, came from ssh. scp was a thin file-copy skin over the top.

That changed in OpenSSH 9.0 (April 2022), which switched scp to use the SFTP protocol by default instead of the legacy rcp-derived one. The old behaviour is still available with `-O` for ancient endpoints, but for anything modern, scp is now effectively an SFTP client with the classic command-line syntax.

The practical consequence either way: everything ssh knows, scp inherits. The same `~/.ssh/config` file, the same keys, the same agent, the same jump hosts and port settings. Most of the "hidden scp features" below are really ssh features that scp gets for free.

The -3 flag

The classic scenario: you are on your laptop, and you need to move a file between two servers that cannot see each other. Different networks, no direct route, no shared credentials.

```
scp -3 host-a:/var/log/app.log host-b:/srv/backup/
```

Historically, without `-3`, your local scp would connect to host-a and instruct the scp process there to write directly to host-b. The data travelled host-a to host-b over their own link: fast, but it required host-a to have network reachability to host-b and credentials to authenticate with, which usually meant private keys sitting on host-a.

With `-3`, the bytes flow host-a to your workstation to host-b. Twice the network traffic and your own uplink becomes the bottleneck, but the two servers never need to know the other exists. Your workstation holds the only credentials, and nothing gets stored on either remote.

Here is the twist. In OpenSSH 8.7 (August 2021), routing remote-to-remote copies through the local host became the default behaviour. On any modern install, `-3` is a no-op, and the old direct mode now lives behind the `-R` flag. OpenSSH 10.6 goes further and deprecates `-R` outright over security concerns. So the feature I had just discovered had already quietly become the standard behaviour years ago. I am not sure whether to feel behind or vindicated.

> ### *Which Behaviour Do You Have?*
>
> *Run `ssh -V` on your workstation. OpenSSH 8.7 or later: remote-to-remote copies already route through your machine, and `-R` restores the old direct mode if you want it. Older than 8.7: you need `-3` for the pass-through behaviour. If a server somewhere is old enough to care, `scp -0` forces the legacy protocol for talking to it.*

Flags I should have known years ago

The full list is in `man scp`, but these are the ones that earn their keep:

- **-P** (capital) sets the port. Lowercase `-p` preserves modification times and mode bits. The capitalisation exists because `-p` was already taken, and it has bitten everyone exactly once.
- **-l** limits bandwidth in Kbit/s. `scp -l 8000 big.iso host:/tmp/` caps the transfer at roughly 1 MB/s. Useful on shared links, and polite in cron jobs.
- **-C** enables compression. A real win for logs, dumps and text over slow links, and a waste of CPU for already-compressed media on fast ones.
- **-r** copies directories recursively. Note that scp follows symlinks during the traversal, which occasionally surprises people copying trees full of them.
- **-B** is batch mode: never prompt for a password or passphrase. Scripts fail cleanly instead of hanging forever on a prompt nobody is watching.
- **-i**, **-o**, **-F** pass through an identity file, an arbitrary ssh option, or an alternate config file. `scp -o ProxyJump=bastion file db:/tmp/` works exactly as you would hope.
- **-J** specifies a jump host directly, with comma-separated hops for deeper chains.
- **-T** disables strict filename checking, a protection added after CVE-2019-6111, where a malicious server could trick the scp client into writing files outside the target directory. Leave it on unless an old script breaks.
- **-O** forces the legacy scp protocol, for embedded gear and old dropbear installs with no SFTP server.
- **-D** connects directly to a local sftp-server binary instead of going over ssh. Mostly a debugging tool.

Let the config file do the work

If you take one thing from this post, make it this: stop typing connection details and put them in `~/.ssh/config`.

```
Host web1
 HostName 203.0.113.10
 User deploy
 Port 2222
 IdentityFile ~/.ssh/id_ed25519_web
```

Then `scp index.php web1:/var/www/` reads like English, and every ssh-family tool (ssh, scp, sftp, rsync, git) picks up the same settings. There is also a URI form, `scp://deploy@203.0.113.10:2222/var/www/`, which is handy when you want the whole destination in one copy-pasteable token.

Key caching: ssh-agent

`ssh-agent` is a small daemon that holds your decrypted private keys in memory. You run `ssh-add ~/.ssh/id_ed25519` once after login, type the passphrase a single time, and every subsequent ssh, scp, sftp and rsync authenticates without asking again. The private key itself never leaves the agent; clients ask the agent to sign challenges rather than reading the key file.

The knobs worth knowing:

```
ssh-add -l # list loaded keys
ssh-add -t 8h ~/.ssh/id_ed25519 # forget the key after 8 hours
ssh-add -c ~/.ssh/id_ed25519 # ask for confirmation on each use
ssh-add -D # remove all keys
```

Add `AddKeysToAgent yes` to your ssh config and keys load themselves into the agent on first use. The `-c` flag looks tedious but matters the moment you start forwarding your agent, which is the next topic.

Chaining hosts: agent forwarding and ProxyJump

This is where it gets fun. Agent forwarding (`ssh -A`, or `ForwardAgent yes` in the config) exposes your local agent to the remote session as a socket. From the bastion, you can ssh onward to a database server using the same key that lives on your workstation. No private key file ever touches the bastion's disk, and you can chain the hops as deep as you like.

```
Host bastion
 HostName bastion.example.com
 ForwardAgent yes
```

The catch: while your session is connected, anyone with root on the bastion can use that socket to authenticate as you, to anywhere your key is accepted. They cannot steal the key, but they can borrow it for exactly as long as you are logged in. The rule of thumb is simple: never forward your agent to a host you would not hand your keys to, and use `ssh-add -c` so every borrowed use at least pops a confirmation on your screen.

The modern alternative is `ProxyJump` (`-J`, or the `ProxyJump` config directive). The connection gets forwarded, not the agent: authentication happens end-to-end between your workstation and the final host, and the bastion just carries encrypted traffic it cannot use. For fixed paths like workstation to bastion to database, ProxyJump is the safer default, and `scp -J bastion file db:/path/` works the same way. Agent forwarding still earns its place for ad-hoc hopping across hosts you control.

A quick word on how good ssh is as a protocol

Step back for a moment and look at what ssh actually is: an authenticated, encrypted transport that multiplexes logical channels over a single TCP connection. Shell sessions and command execution are just two channel types. The same connection can carry SFTP, forwarded agent sockets, X11, local and remote port forwards, a dynamic forward that turns ssh into a SOCKS proxy, and with `-w` a full layer-3 VPN between tun devices. Add `ControlMaster` and `ControlPersist` and your subsequent sessions ride over the existing connection with no re-authentication, which makes repeated scp calls effectively instant. Even the escape sequences are worth knowing: on a new line, `~.` kills a hung session and `~C` opens a command line to add port forwards mid-session.

There is a very good visual guide to the tunnelling side of this that I saved to the gav.ilicio.us archive a while back: A Practical Guide to SSH Tunnels. It covers local, remote and dynamic forwarding with clear diagrams, and it is the single best read on the topic I have found.

And this is why production servers should say no

Everything above is a feature on your own machines and a liability on a production server. Each one is a pivot primitive: port forwarding turns a compromised account into a tunnel straight through your firewall policy, agent forwarding exposes your credentials to anyone with root on the box, and tun devices hand out network interfaces. The convenience and the attack surface are the same thing viewed from different ends.

A reasonable baseline for a production `sshd_config`:

```
PermitRootLogin no
PasswordAuthentication no
AllowAgentForwarding no
AllowTcpForwarding no
X11Forwarding no
PermitTunnel no
GatewayPorts no
LogLevel VERBOSE
```

Two notes on that list. If a server genuinely needs TCP forwarding for something, scope it with `PermitOpen host:port` so it can only reach the one destination you intended, rather than `AllowTcpForwarding yes`. And `LogLevel VERBOSE` logs the key fingerprint used at login, which is what you want when you are later asking who connected with which key.

For accounts that exist purely to receive files, such as backup targets, do not give them a shell at all:

```
Match Group sftp-only
 ForceCommand internal-sftp
 ChrootDirectory /srv/sftp/%u
 AllowTcpForwarding no
 X11Forwarding no
```

That gives you a working scp and sftp endpoint (SFTP-based scp included, another reason the OpenSSH 9.0 change matters) with no shell, no forwarding, and the account locked into its own directory tree.

Closing

Thirty years in, and the man page still has things to teach me. The broader lesson is that scp was never a small tool with a short man page; it is the entire ssh stack wearing a copy command as a hat, and the ssh stack is deep. Next weekend's project is reading `ssh_config(5)` front to back. There is

probably another `-3` in there somewhere.

Related:

- A Practical Guide to SSH Tunnels - the visual guide to local, remote and dynamic forwarding, saved in the link archive
- gav.ilicio.us - the rest of the saved links
- OpenSSH release notes - where the -3, -R and SFTP-default changes are recorded

---

Downloaded from https://www.gavinj.net/post/scp-tips-and-tricks
Generated November 14, 2026. Copyright Gavin Jackson. All rights reserved.