NAME

sshd — OpenSSH SSH daemon

SYNOPSIS

sshd [**-46DdeiqTt**] [**-b** *bits*] [**-C** *connection_spec*] [**-c** *host_certificate_file*]
[**-E** *log_file*] [**-f** *config_file*] [**-g** *login_grace_time*]
[**-h** *host_key_file*] [**-k** *key_gen_time*] [**-o** *option*] [**-p** *port*] [**-u** *len*]

DESCRIPTION

sshd (OpenSSH Daemon) is the daemon program for ssh(1). Together these programs replace rlogin and rsh, and provide secure encrypted communications between two untrusted hosts over an insecure network.

sshd listens for connections from clients. It is normally started at boot from `/etc/init.d/ssh` (or `/etc/init/ssh.conf` on systems using the Upstart init daemon). It forks a new daemon for each incoming connection. The forked daemons handle key exchange, encryption, authentication, command execution, and data exchange.

sshd can be configured using command-line options or a configuration file (by default ; -- sshd_config(5)) command-line options override values specified in the configuration file. **sshd** rereads its configuration file when it receives a hangup signal, `SIGHUP`, by executing itself with the name and options it was started with, e.g. `/usr/sbin/sshd`.

The options are as follows:

-4 Forces **sshd** to use IPv4 addresses only.

-6 Forces **sshd** to use IPv6 addresses only.

-b *bits*
Specifies the number of bits in the ephemeral protocol version 1 server key (default 1024).

-C *connection_spec*
Specify the connection parameters to use for the **-T** extended test mode. If provided, any **Match** directives in the configuration file that would apply to the specified user, host, and address will be set before the configuration is written to standard output. The connection parameters are supplied as keyword=value pairs. The keywords are "user", "host", "laddr", "lport", and "addr". All are required and may be supplied in any order, either with multiple **-C** options or as a comma-separated list.

-c *host_certificate_file*
Specifies a path to a certificate file to identify **sshd** during key exchange. The certificate file must match a host key file specified using the **-h** option or the **HostKey** configuration directive.

-D When this option is specified, **sshd** will not detach and does not become a daemon. This allows easy monitoring of **sshd**.

-d Debug mode. The server sends verbose debug output to standard error, and does not put itself in the background. The server also will not fork and will only process one connection. This option is only intended for debugging for the server. Multiple **-d** options increase the debugging level. Maximum is 3.

-E *log_file*
Append debug logs to *log_file* instead of the system log.

-e Write debug logs to standard error instead of the system log.

-f *config_file*
Specifies the name of the configuration file. The default is `/etc/ssh/sshd_config`. **sshd** refuses to start if there is no configuration file.

-g *login_grace_time*
Gives the grace time for clients to authenticate themselves (default 120 seconds). If the client fails to authenticate the user within this many seconds, the server disconnects and exits. A value of zero indicates no limit.

-h *host_key_file*
Specifies a file from which a host key is read. This option must be given if **sshd** is not run as root (as the normal host key files are normally not readable by anyone but root). The default is `/etc/ssh/ssh_host_key` for protocol version 1, and `/etc/ssh/ssh_host_dsa_key`, `/etc/ssh/ssh_host_ecdsa_key`. `/etc/ssh/ssh_host_ed25519_key` and `/etc/ssh/ssh_host_rsa_key` for protocol version 2. It is possible to have multiple host key files for the different protocol versions and host key algorithms.

-i Specifies that **sshd** is being run from inetd(8). **sshd** is normally not run from inetd because it needs to generate the server key before it can respond to the client, and this may take tens of seconds. Clients would have to wait too long if the key was regenerated every time. However, with small key sizes (e.g. 512) using **sshd** from inetd may be feasible.

-k *key_gen_time*
Specifies how often the ephemeral protocol version 1 server key is regenerated (default 3600 seconds, or one hour). The motivation for regenerating the key fairly often is that the key is not stored anywhere, and after about an hour it becomes impossible to recover the key for decrypting intercepted communications even if the machine is cracked into or physically seized. A value of zero indicates that the key will never be regenerated.

-o *option*
Can be used to give options in the format used in the configuration file. This is useful for specifying options for which there is no separate command-line flag. For full details of the options, and their values, see sshd_config(5).

-p *port*
Specifies the port on which the server listens for connections (default 22). Multiple port options are permitted. Ports specified in the configuration file with the**Port** option are ignored when a command-line port is specified. Ports specified using the **ListenAddress** option override command-line ports.

-q Quiet mode. Nothing is sent to the system log. Normally the beginning, authentication, and termination of each connection is logged.

-T Extended test mode. Check the validity of the configuration file, output the effective configuration to stdout and then exit. Optionally, **Match** rules may be applied by specifying the connection parameters using one or more **-C** options.

-t Test mode. Only check the validity of the configuration file and sanity of the keys. This is useful for updating **sshd** reliably as configuration options may change.

-u *len*
This option is used to specify the size of the field in the `utmp` structure that holds the remote host name. If the resolved host name is longer than *len*, the dotted decimal value will be used instead. This allows hosts with very long host names that overflow this field to still be uniquely identified. Specifying **-u0** indicates that only dotted decimal addresses should be put into the `utmp` file. **-u0** may also be used to prevent **sshd** from making DNS requests unless the authentication mechanism or configuration requires it. Authentication mechanisms that may require DNS include

RhostsRSAAuthentication, **HostbasedAuthentication**, and using a **from="pattern-list"** option in a key file. Configuration options that require DNS include using a USER@HOST pattern in **AllowUsers** or **DenyUsers**.

AUTHENTICATION

The OpenSSH SSH daemon supports SSH protocols 1 and 2. The default is to use protocol 2 only, though this can be changed via the **Protocol** option in sshd_config(5). Protocol 2 supports DSA, ECDSA, ED25519 and RSA keys; protocol 1 only supports RSA keys. For both protocols, each host has a host-specific key, normally 2048 bits, used to identify the host.

Forward security for protocol 1 is provided through an additional server key, normally 768 bits, generated when the server starts. This key is normally regenerated every hour if it has been used, and is never stored on disk. Whenever a client connects, the daemon responds with its public host and server keys. The client compares the RSA host key against its own database to verify that it has not changed. The client then generates a 256-bit random number. It encrypts this random number using both the host key and the server key, and sends the encrypted number to the server. Both sides then use this random number as a session key which is used to encrypt all further communications in the session. The rest of the session is encrypted using a conventional cipher, currently Blowfish or 3DES, with 3DES being used by default. The client selects the encryption algorithm to use from those offered by the server.

For protocol 2, forward security is provided through a Diffie-Hellman key agreement. This key agreement results in a shared session key. The rest of the session is encrypted using a symmetric cipher, currently 128-bit AES, Blowfish, 3DES, CAST128, Arcfour, 192-bit AES, or 256-bit AES. The client selects the encryption algorithm to use from those offered by the server. Additionally, session integrity is provided through a cryptographic message authentication code (hmac-md5, hmac-sha1, umac-64, umac-128, hmac-ripemd160, hmac-sha2-256 or hmac-sha2-512).

Finally, the server and the client enter an authentication dialog. The client tries to authenticate itself using host-based authentication, public key authentication, challenge-response authentication, or password authentication.

Regardless of the authentication type, the account is checked to ensure that it is accessible. An account is not accessible if it is locked, listed in **DenyUsers** or its group is listed in **DenyGroups** . The definition of a locked account is system dependant. Some platforms have their own account database (eg AIX) and some modify the passwd field (∗LK∗ on Solaris and UnixWare, '∗' on HP-UX, containing`Nologin` on T ru64, a leading ∗LOCKED∗ on FreeBSD and a leading '!' on most Linuxes). If there is a requirement to disable password authentication for the account while allowing still public-key, then the passwd field should be set to something other than these values (eg 'NP' or∗NP∗).

If the client successfully authenticates itself, a dialog for preparing the session is entered. At this time the client may request things like allocating a pseudo-tty, forwarding X11 connections, forwarding TCP connections, or forwarding the authentication agent connection over the secure channel.

After this, the client either requests a shell or execution of a command. The sides then enter session mode. In this mode, either side may send data at any time, and such data is forwarded to/from the shell or command on the server side, and the user terminal in the client side.

When the user program terminates and all forwarded X11 and other connections have been closed, the server sends command exit status to the client, and both sides exit.

LOGIN PROCESS

When a user successfully logs in, **sshd** does the following:

1. If the login is on a tty, and no command has been specified, prints last login time and `/etc/motd` (unless prevented in the configuration file or by `~/.hushlogin`; see the FILES section).

2. If the login is on a tty, records login time.
3. Checks `/etc/nologin`; if it exists, prints contents and quits (unless root).
4. Changes to run with normal user privileges.
5. Sets up basic environment.
6. Reads the file `~/.ssh/environment`, if it exists, and users are allowed to change their environment. See the**`PermitUserEnvironment`** option in sshd_config(5).
7. Changes to user's home directory.
8. If `~/.ssh/rc` exists and the `sshd_config`(5) **`PermitUserRC`** option is set, runs it; else if `/etc/ssh/sshrc` exists, runs it; otherwise runs xauth. The "rc" files are given the X11 authentication protocol and cookie in standard input. See SSHRC, below.
9. Runs user's shell or command.

SSHRC

If the file `~/.ssh/rc` exists, `sh`(1) runs it after reading the environment files but before starting the user's shell or command. It must not produce any output on stdout; stderr must be used instead. If X11 forwarding is in use, it will receive the "proto cookie" pair in its standard input (and `DISPLAY` in its environment). The script must call `xauth`(1) because **`sshd`** will not run xauth automatically to add X11 cookies.

The primary purpose of this file is to run any initialization routines which may be needed before the user's home directory becomes accessible; AFS is a particular example of such an environment.

This file will probably contain some initialization code followed by something similar to:

```
if read proto cookie && [ -n "$DISPLAY" ]; then
if [ ‘echo $DISPLAY | cut -c1-10‘ = ’localhost:’ ]; then
# X11UseLocalhost=yes
echo add unix:‘echo $DISPLAY |
cut -c11-‘ $proto $cookie
else
# X11UseLocalhost=no
echo add $DISPLAY $proto $cookie
fi | xauth -q -
fi
```

If this file does not exist, `/etc/ssh/sshrc` is run, and if that does not exist either, xauth is used to add the cookie.

AUTHORIZED_KEYS FILE FORMAT

`AuthorizedKeysFile` specifies the files containing public keys for public key authentication; if none is specified, the default is `~/.ssh/authorized_keys` and `~/.ssh/authorized_keys2`. Each line of the file contains one key (empty lines and lines starting with a '#' are ignored as comments). Protocol 1 public keys consist of the following space-separated fields: options, bits, exponent, modulus, comment. Protocol 2 public key consist of: options, keytype, base64-encoded key, comment. The options field is optional; its presence is determined by whether the line starts with a number or not (the options field never starts with a number). The bits, exponent, modulus, and comment fields give the RSA key for protocol version 1; the comment field is not used for anything (but may be convenient for the user to identify the key). For protocol version 2 the keytype is "ecdsa-sha2-nistp256", "ecdsa-sha2-nistp384", "ecdsa-sha2-nistp521", "ssh-ed25519", "ssh-dss" or "ssh-rsa".

Note that lines in this file are usually several hundred bytes long (because of the size of the public key encoding) up to a limit of 8 kilobytes, which permits DSA keys up to 8 kilobits and RSA keys up to 16 kilobits. You don't want to type them in; instead, copy the `identity.pub`, `id_dsa.pub`, `id_ecdsa.pub`, `id_ed25519.pub`, or the `id_rsa.pub` file and edit it.

sshd enforces a minimum RSA key modulus size for protocol 1 and protocol 2 keys of 768 bits.

The options (if present) consist of comma-separated option specifications. No spaces are permitted, except within double quotes. The following option specifications are supported (note that option keywords are case-insensitive):

cert-authority
: Specifies that the listed key is a certification authority (CA) that is trusted to validate signed certificates for user authentication.

  Certificates may encode access restrictions similar to these key options. If both certificate restrictions and key options are present, the most restrictive union of the two is applied.

command="command"
: Specifies that the command is executed whenever this key is used for authentication. The command supplied by the user (if any) is ignored. The command is run on a pty if the client requests a pty; otherwise it is run without a tty. If an 8-bit clean channel is required, one must not request a pty or should specify **no-pty**. A quote may be included in the command by quoting it with a backslash. This option might be useful to restrict certain public keys to perform just a specific operation. An example might be a key that permits remote backups but nothing else. Note that the client may specify TCP and/or X11 forwarding unless they are explicitly prohibited. The command originally supplied by the client is available in the `SSH_ORIGINAL_COMMAND` environment variable. Note that this option applies to shell, command or subsystem execution. Also note that this command may be superseded by either a `sshd_config`(5) **ForceCommand** directive or a command embedded in a certificate.

environment="NAME=value"
: Specifies that the string is to be added to the environment when logging in using this key. Environment variables set this way override other default environment values. Multiple options of this type are permitted. Environment processing is disabled by default and is controlled via the **PermitUserEnvironment** option. This option is automatically disabled if **UseLogin** is enabled.

from="pattern-list"
: Specifies that in addition to public key authentication, either the canonical name of the remote host or its IP address must be present in the comma-separated list of patterns. See PATTERNS in `ssh_config`(5) for more information on patterns.

  In addition to the wildcard matching that may be applied to hostnames or addresses, a **from** stanza may match IP addresses using CIDR address/masklen notation.

  The purpose of this option is to optionally increase security: public key authentication by itself does not trust the network or name servers or anything (but the key); however, if somebody somehow steals the key, the key permits an intruder to log in from anywhere in the world. This additional option makes using a stolen key more difficult (name servers and/or routers would have to be compromised in addition to just the key).

no-agent-forwarding
: Forbids authentication agent forwarding when this key is used for authentication.

no-port-forwarding
Forbids TCP forwarding when this key is used for authentication. Any port forward requests by the client will return an error. This might be used, e.g. in connection with the **command** option.

no-pty
Prevents tty allocation (a request to allocate a pty will fail).

no-user-rc
Disables execution of ~/.ssh/rc.

no-X11-forwarding
Forbids X11 forwarding when this key is used for authentication. Any X11 forward requests by the client will return an error.

permitopen="host:port"
Limit local ''ssh -L'' port forwarding such that it may only connect to the specified host and port. IPv6 addresses can be specified by enclosing the address in square brackets. Multiple **permitopen** options may be applied separated by commas. No pattern matching is performed on the specified hostnames, they must be literal domains or addresses. A port specification of * matches any port.

principals="principals"
On a **cert-authority** line, specifies allowed principals for certificate authentication as a comma-separated list. At least one name from the list must appear in the certificate's list of principals for the certificate to be accepted. This option is ignored for keys that are not marked as trusted certificate signers using the **cert-authority** option.

tunnel="n"
Force a tun(4) device on the server. Without this option, the next available device will be used if the client requests a tunnel.

An example authorized_keys file:

```
# Comments allowed at start of line
ssh-rsa AAAAB3Nza...LiPk== user@example.net
from="*.sales.example.net,!pc.sales.example.net" ssh-rsa
AAAAB2...19Q== john@example.net
command="dump /home",no-pty,no-port-forwarding ssh-dss
AAAAC3...51R== example.net
permitopen="192.0.2.1:80",permitopen="192.0.2.2:25" ssh-dss
AAAAB5...21S==
tunnel="0",command="sh /etc/netstart tun0" ssh-rsa AAAA...==
jane@example.net
```

SSH_KNOWN_HOSTS FILE FORMAT

The /etc/ssh/ssh_known_hosts and ~/.ssh/known_hosts files contain host public keys for all known hosts. The global file should be prepared by the administrator (optional), and the per-user file is maintained automatically: whenever the user connects from an unknown host, its key is added to the per-user file.

Each line in these files contains the following fields: markers (optional), hostnames, bits, exponent, modulus, comment. The fields are separated by spaces.

The marker is optional, but if it is present then it must be one of "@cert-authority", to indicate that the line contains a certification authority (CA) key, or "@revoked", to indicate that the key contained on the line is revoked and must not ever be accepted. Only one marker should be used on a key line.

Hostnames is a comma-separated list of patterns ('*' and '?' act as wildcards); each pattern in turn is matched against the canonical host name (when authenticating a client) or against the user-supplied name (when authenticating a server). A pattern may also be preceded by '!' to indicate negation: if the host name matches a negated pattern, it is not accepted (by that line) even if it matched another pattern on the line. A hostname or address may optionally be enclosed within '[' and ']' brackets then followed by ':' and a non-standard port number.

Alternately, hostnames may be stored in a hashed form which hides host names and addresses should the file's contents be disclosed. Hashed hostnames start with a '|' character. Only one hashed hostname may appear on a single line and none of the above negation or wildcard operators may be applied.

Bits, exponent, and modulus are taken directly from the RSA host key; they can be obtained, for example, from `/etc/ssh/ssh_host_key.pub`. The optional comment field continues to the end of the line, and is not used.

Lines starting with '#' and empty lines are ignored as comments.

When performing host authentication, authentication is accepted if any matching line has the proper key; either one that matches exactly or, if the server has presented a certificate for authentication, the key of the certification authority that signed the certificate. For a key to be trusted as a certification authority, it must use the "@cert-authority" marker described above.

The known hosts file also provides a facility to mark keys as revoked, for example when it is known that the associated private key has been stolen. Revoked keys are specified by including the "@revoked" marker at the beginning of the key line, and are never accepted for authentication or as certification authorities, but instead will produce a warning from `ssh`(1) when they are encountered.

It is permissible (but not recommended) to have several lines or different host keys for the same names. This will inevitably happen when short forms of host names from different domains are put in the file. It is possible that the files contain conflicting information; authentication is accepted if valid information can be found from either file.

Note that the lines in these files are typically hundreds of characters long, and you definitely don't want to type in the host keys by hand. Rather, generate them by a script, `ssh-keyscan`(1) or by taking `/etc/ssh/ssh_host_key.pub` and adding the host names at the front. `ssh-keygen`(1) also offers some basic automated editing for `~/.ssh/known_hosts` including removing hosts matching a host name and converting all host names to their hashed representations.

An example ssh_known_hosts file:

```
# Comments allowed at start of line
closenet,...,192.0.2.53 1024 37 159...93 closenet.example.net
cvs.example.net,192.0.2.10 ssh-rsa AAAA1234.....=
# A hashed hostname
|1|JfKTdBh7rNbXkVAQCRp4OQoPfmI=|USECr3SWf1JUPsms5AqfD5QfxkM= ssh-rsa
AAAA1234.....=
# A revoked key
@revoked * ssh-rsa AAAAB5W...
# A CA key, accepted for any host in *.mydomain.com or *.mydomain.org
@cert-authority *.mydomain.org,*.mydomain.com ssh-rsa AAAAB5W...
```

FILES

`~/.hushlogin`

This file is used to suppress printing the last login time and `/etc/motd`, if **`PrintLastLog`** and **`PrintMotd`**, respectively, are enabled. It does not suppress printing of the banner specified by **`Banner`**.

`~/.rhosts`
: This file is used for host-based authentication (see `ssh`(1) for more information). On some machines this file may need to be world-readable if the user's home directory is on an NFS partition, because **`sshd`** reads it as root. Additionally, this file must be owned by the user, and must not have write permissions for anyone else. The recommended permission for most machines is read/write for the user, and not accessible by others.

`~/.shosts`
: This file is used in exactly the same way as `.rhosts`, but allows host-based authentication without permitting login with rlogin/rsh.

`~/.ssh/`
: This directory is the default location for all user-specific configuration and authentication information. There is no general requirement to keep the entire contents of this directory secret, but the recommended permissions are read/write/execute for the user, and not accessible by others.

`~/.ssh/authorized_keys`
: Lists the public keys (DSA, ECDSA, ED25519, RSA) that can be used for logging in as this user. The format of this file is described above. The content of the file is not highly sensitive, but the recommended permissions are read/write for the user, and not accessible by others.

  If this file, the `~/.ssh` directory, or the user's home directory are writable by other users, then the file could be modified or replaced by unauthorized users. In this case, **`sshd`** will not allow it to be used unless the **`StrictModes`** option has been set to "no".

`~/.ssh/environment`
: This file is read into the environment at login (if it exists). It can only contain empty lines, comment lines (that start with '#'), and assignment lines of the form name=value. The file should be writable only by the user; it need not be readable by anyone else. Environment processing is disabled by default and is controlled via the **`PermitUserEnvironment`** option.

`~/.ssh/known_hosts`
: Contains a list of host keys for all hosts the user has logged into that are not already in the systemwide list of known host keys. The format of this file is described above. This file should be writable only by root/the owner and can, but need not be, world-readable.

`~/.ssh/rc`
: Contains initialization routines to be run before the user's home directory becomes accessible. This file should be writable only by the user, and need not be readable by anyone else.

`/etc/hosts.allow`
`/etc/hosts.deny`
: Access controls that should be enforced by tcp-wrappers are defined here. Further details are described in hosts_access(5).

`/etc/hosts.equiv`
: This file is for host-based authentication (see . -- ssh(1)) It should only be writable by root.

`/etc/ssh/moduli`
: Contains Diffie-Hellman groups used for the "Diffie-Hellman Group Exchange". The file format is described in moduli(5).

`/etc/motd`
: See motd(5).

`/etc/nologin`
: If this file exists, **sshd** refuses to let anyone except root log in. The contents of the file are displayed to anyone trying to log in, and non-root connections are refused. The file should be world-readable.

`/etc/ssh/shosts.equiv`
: This file is used in exactly the same way as `hosts.equiv`, but allows host-based authentication without permitting login with rlogin/rsh.

`/etc/ssh/ssh_host_key`
`/etc/ssh/ssh_host_dsa_key`
`/etc/ssh/ssh_host_ecdsa_key`
`/etc/ssh/ssh_host_ed25519_key`
`/etc/ssh/ssh_host_rsa_key`
: These files contain the private parts of the host keys. These files should only be owned by root, readable only by root, and not accessible to others. Note that **sshd** does not start if these files are group/world-accessible.

`/etc/ssh/ssh_host_key.pub`
`/etc/ssh/ssh_host_dsa_key.pub`
`/etc/ssh/ssh_host_ecdsa_key.pub`
`/etc/ssh/ssh_host_ed25519_key.pub`
`/etc/ssh/ssh_host_rsa_key.pub`
: These files contain the public parts of the host keys. These files should be world-readable but writable only by root. Their contents should match the respective private parts. These files are not really used for anything; they are provided for the convenience of the user so their contents can be copied to known hosts files. These files are created using `ssh-keygen`(1).

`/etc/ssh/ssh_known_hosts`
: Systemwide list of known host keys. This file should be prepared by the system administrator to contain the public host keys of all machines in the organization. The format of this file is described above. This file should be writable only by root/the owner and should be world-readable.

`/etc/ssh/sshd_config`
: Contains configuration data for **sshd**. The file format and configuration options are described in sshd_config(5).

`/etc/ssh/sshrc`
: Similar to `~/.ssh/rc`, it can be used to specify machine-specific login-time initializations globally. This file should be writable only by root, and should be world-readable.

`/var/run/sshd`
: `chroot`(2) directory used by **sshd** during privilege separation in the pre-authentication phase. The directory should not contain any files and must be owned by root and not group or world-writable.

`/var/run/sshd.pid`
: Contains the process ID of the **sshd** listening for connections (if there are several daemons running concurrently for different ports, this contains the process ID of the one started last). The content of this file is not sensitive; it can be world-readable.

SEE ALSO

scp(1), sftp(1), ssh(1), `ssh-add`(1), `ssh-agent`(1), `ssh-keygen`(1), `ssh-keyscan`(1), chroot(2), hosts_access(5), moduli(5), sshd_config(5), inetd(8), `sftp-server`(8)

AUTHORS

OpenSSH is a derivative of the original and free ssh 1.2.12 release by Tatu Ylonen. Aaron Campbell, Bob Beck, Markus Friedl, Niels Provos, Theo de Raadt and Dug Song removed many bugs, re-added newer features and created OpenSSH. Markus Friedl contributed the support for SSH protocol versions 1.5 and 2.0. Niels Provos and Markus Friedl contributed support for privilege separation.