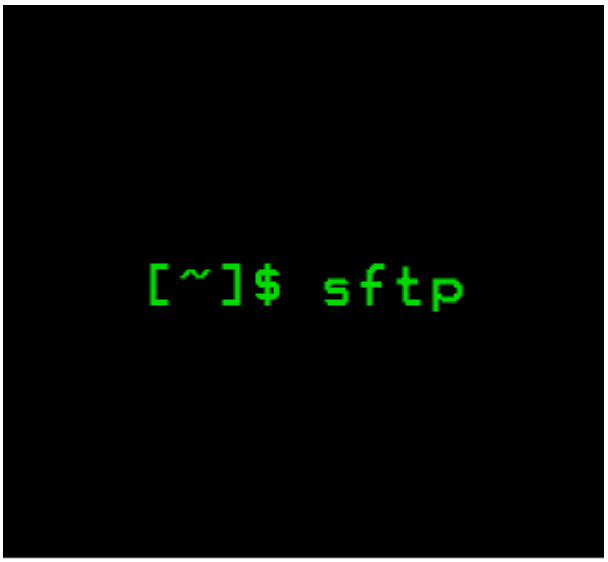
https://www.computerhope.com/unix/sftp.htm

Linux sftp command help and examples

Computer Hope

Updated: 05/04/2019 by



On Unix-like operating systems, **sftp** is the command-line interface for using the SFTP secure file transfer protocol. It is an encrypted version of FTP. It transfers files securely over a network connection.

Description

You may already be familiar with FTP: it's a very simple, and very *insecure* method for uploading or downloading files over a network connection. It does not provide any sort of secure encryption in the session or in the data transfer.

sftp provides this functionality. Think of it as an encrypted version of **ftp**.

Note

If you need to transfer files over *anonymous* FTP, **sftp** is *not* the program to use. Because all **sftp** connections are encrypted, they require a username and password (or public key authentication). So, for anonymous FTP transfers, use regular **ftp**.

Syntax

sftp performs all operations over an encrypted **ssh** session. It uses many of the features of **ssh**, such as public key authentication and data compression.

There are four basic ways to use **sftp**, and the command syntax for each is listed here. (For more information about each option and its possible values, see the Options section, below).

1. The first is an *interactive session*. In this mode, **sftp** connects and logs into the specified host, then enters its interactive command mode, where you type all your commands at a prompt. To launch an interactive session of **sftp**, use the following syntax:

   ```
   sftp [-1246Cpqrv] [-B buffer_size] [-b batchfile] [-c cipher]  [-D
   sftp_server_path] [-F ssh_config] [-i identity_file]  [-l limit] [-o
   ssh_option] [-P port] [-R num_requests]  [-S program] [-s subsystem |
   sftp_server] host
   ```

   See Interactive Mode for an example of using **sftp** this way.
2. You can also use **sftp** to retrieve files automatically, without any prompted interaction:

   ```
   sftp [user@]host[:file ...]
   ```

   See Automatic Retrieval Mode for an example of using **sftp** in this way.
3. Or you can tell **sftp** to start its interactive session in a specific remote directory:

   ```
   sftp [user@]host[:dir[/]]
   ```

   See Starting Interactive Mode In A Specific Remote Directory for an example of using **sftp** this way.
4. Lastly, you can run a completely automated session using the **-b** option. The "b" stands for "batch mode."

   To use batch mode, it is necessary to configure non-interactive authentication, such as public key authentication, so that you don't have to manually enter a password. The **sftp** syntax for this mode is:

   ```
   sftp -b batchfile [user@]host
   ```

   For examples of using batch mode, and a guide to setting up public key authentication, see Batch Mode.

Options

Here is a description of each of the options listed in the command syntaxes listed above.

-1 Specify the use of protocol version 1, which dates back to 1997. This option provides compatibility with very old servers. If you're not sure that you need it, do not specify this option.

-2	Specify the use of protocol version 2, which dates back to 1997. This option provides compatibility with very old servers. If you're not sure that you need it, do not specify this option.
-4	Forces **sftp** to use IPv4 addresses only.
-6	Forces **sftp** to use IPv6 addresses only.
-B *buffer_size*	Specify the size of the buffer that **sftp** uses when transferring files. Larger buffers require fewer round trips, but require more memory to do so.. The default is **32768** bytes.
-b *batchfile*	Batch mode reads a series of commands from an input batchfile instead of stdin. Since it lacks user interaction it should be used in conjunction with non-interactive authentication. If *batchfile* is specified as a dash ("**-**"), standard input will be read for the batch of commands. **sftp** will abort if any of the following commands fail: **get**, **put**, **rename**, **ln**, **rm**, **mkdir**, **chdir**, **ls**, **lchdir**, **chmod**, **chown**, **chgrp**, **lpwd**, **df**, **symlink**, and **lmkdir**. Termination on error can be suppressed on a command by command basis by prefixing the command with a "**-**" character (for example, **-rm /tmp/file***).
-C	Enables compression (via **ssh**'s **-C** flag).
-c *cipher*	Selects the cipher to use for encrypting the data transfers. This option is directly passed to **ssh**.
-D *sftp_server_path*	Connect directly to a local **sftp** server (rather than via **ssh**). This option may be useful in debugging the client and server.
-F *ssh_config*	Specifies an alternative per-user configuration file for **ssh**. This option is directly passed to **ssh**.
-i *identity_file*	Selects the file from which the identity (private key) for public key authentication is read. This option is directly passed to **ssh**.
-l *limit*	Limits the used bandwidth, specified in Kbit/s.
-o *ssh_option*	Can be used to pass options to **ssh** in the format used in **ssh_config**. This is useful for specifying options for which there is no separate **sftp** command-line flag. For example, to specify an alternate port use: **sftp -oPort=24**.

AddressFamily	Specifies which address family to use when connecting. Valid arguments are "**any**", "**inet**" (use IPv4 only), or "**inet6**" (use IPv6 only).
BatchMode	If set to "**yes**", passphrase/password querying will be disabled. Also, the **ServerAliveInterval** option will be set to **300** seconds by default. This option is useful in scripts and other batch jobs where no user is present to supply the password, and where it is desirable to detect a broken network swiftly. The argument must be "**yes**" or "**no**". The default is "**no**".
BindAddress	Use the specified address on the local machine as the source address of the connection. Only useful on systems with more than one address. Note that this option does not work if **UsePrivilegedPort** is set to "**yes**".

ChallengeResponseAuthentication	Specifies whether to use "challenge-response" authentication. The argument to this keyword must be "**yes**" or "**no**". The default is "**yes**".
CheckHostIP	If this flag is set to "**yes**", **ssh** will additionally check the host IP address in the **known_hosts** file. This allows **ssh** to detect if a host key changed due to DNS spoofing. If the option is set to "**no**", the check will not be executed. The default is "**yes**".
Cipher	Specifies the cipher to use for encrypting the session in protocol version 1. Currently, "**blowfish**", "**3des**", and "**des**" are supported. **des** is only supported in the **ssh** client for interoperability with legacy protocol 1 implementations that do not support the **3des** cipher. Its use is strongly discouraged due to cryptographic weaknesses. The default is "**3des**".
Ciphers	Specifies the ciphers allowed for protocol version 2 in order of preference. Multiple ciphers must be comma-separated. The supported ciphers are "**3des-cbc**", "**aes128-cbc**", "**aes192-cbc**", "**aes256-cbc**", "**aes128-ctr**", "**aes192-ctr**", "**aes256-ctr**", "**arcfour128**", "**arcfour256**", "**arcfour**", "**blowfish-cbc**", and "**cast128-cbc**". The default value is the following really long string: **aes128-ctr,aes192-ctr,aes256-ctr,arcfour256,arcfour128,aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,aes192-cbc,aes256-cbc,arcfour**
Compression	Specifies whether to use compression. The argument must be "**yes**" or "**no**". The default is "**no**".
CompressionLevel	Specifies the compression level to use if compression is enabled. The argument must be an integer from **1** (fast) to **9** (slower, but best compression ratio). The default level is **6**, which is good for most applications. The meaning of the values is the same as in **gzip**. Note that this option applies to protocol version 1 only.
ConnectionAttempts	Specifies the number of tries (one per second) to make before exiting. The argument must be an integer. This may be useful in scripts if the connection sometimes fails. The default is **1**.
ConnectTimeout	Specifies the timeout (in seconds) used when

connecting to the SSH server, instead of using the default system TCP timeout. This value is used only when the target is down or really unreachable, not when it refuses the connection.

ControlMaster

Enables the sharing of multiple sessions over a single network connection. When set to "**yes**", **ssh** will listen for connections on a control socket specified using the **ControlPath** argument. Additional sessions can connect to this socket using the same **ControlPath** with **ControlMaster** set to "**no**" (the default). These sessions will try to reuse the master instance's network connection rather than initiating new ones, but will fall back to connecting normally if the control socket does not exist, or is not listening.

Setting this to "**ask**" will cause **ssh** to listen for control connections, but require confirmation using the **SSH_ASKPASS** program before they are accepted. If the **ControlPath** cannot be opened, **ssh** will continue without connecting to a master instance.

X11 and **ssh-agent** forwarding is supported over these multiplexed connections, however the display and agent forwarded will be the one belonging to the master connection i.e. it is not possible to forward multiple displays or agents.

Two additional options allow for opportunistic multiplexing: try to use a master connection but fall back to creating a new one if one does not already exist. These options are: "**auto**" and "**autoask**". The latter requires confirmation like the "**ask**" option.

ControlPath

Specify the path to the control socket used for connection sharing as described in the **ControlMaster** section above or the string "**none**" to disable connection sharing. In the path, '**%L**' will be substituted by the first component of the local hostname, '**%l**' will be substituted by the local hostname (including any domain name), '**%h**' will be substituted by the target hostname, '**%n**' will be substituted by the original target hostname specified on the command line, '**%p**' the port, '**%r**' by the remote login username, and '**%u**' by the username of the user running **ssh**. It is

	recommended that any **ControlPath** used for opportunistic connection sharing include at least **%h**, **%p**, and **%r**. This ensures that shared connections are uniquely identified.
ControlPersist	When used in conjunction with **ControlMaster**, specifies that the master connection should remain open in the background (waiting for future client connections) after the initial client connection has been closed. If set to "**no**", then the master connection will not be placed into the background, and will close as soon as the initial client connection is closed. If set to "**yes**", then the master connection will remain in the background indefinitely (until killed or closed via a mechanism such as the **ssh** option "**-O exit**"). If set to a time in seconds, or a time in any of the formats documented in **sshd_config**, then the backgrounded master connection will automatically terminate after it has remained idle (with no client connections) for the specified time.
GlobalKnownHostsFile	Specifies one or more files to use for the global host key database, separated by whitespace. The default is **/etc/ssh/ssh_known_hosts**, **/etc/ssh/ssh_known_hosts2**.
GSSAPIAuthentication	Specifies whether user authentication based on GSSAPI (the Generic Security Service Application Program Interface) is allowed. The default is "**no**". Note that this option applies to protocol version 2 only.
GSSAPIDelegateCredentials	Forward (delegate) credentials to the server. The default is "**no**". Note that this option applies to protocol version 2 connections using GSSAPI.
HashKnownHosts	Indicates that **ssh** should hash hostnames and addresses when they are added to **~/.ssh/known_hosts**. These hashed names may be used normally by **ssh** and **sshd**, but they do not reveal identifying information should the file's contents be disclosed. The default is "**no**". Note that existing names and addresses in known hosts files will not be converted automatically, but may be manually hashed using the **ssh-keygen** key generator. Use of this option may break facilities such as tab-completion that rely on being able to read unhashed hostnames from **~/.ssh/known_hosts**.

Host	Restricts the following declarations (up to the next **Host** keyword) to be only for those hosts that match one of the patterns given after the keyword. If more than one pattern is provided, they should be separated by whitespace. A single "*****" as a pattern can be used to provide global defaults for all hosts. The host is the hostname argument given on the command line (i.e. the name is not converted to a canonicalized hostname before matching). A pattern entry may be negated by prefixing it with an exclamation mark ("**!**"). If a negated entry is matched, then the Host entry is ignored, regardless of whether any other patterns on the line match. Negated matches are therefore useful to provide exceptions for wildcard matches.
HostbasedAuthentication	Specifies whether to try **rhosts**-based authentication with public key authentication. The argument must be "**yes**" or "**no**". The default is "**no**". This option applies to protocol version 2 only and is similar to **RhostsRSAAuthentication**.
HostKeyAlgorithms	Specifies the protocol version 2 host key algorithms that the client wants to use in order of preference. The default for this option is the following really, really long string: **ecdsa-sha2-nistp256-cert-v01@openssh.com, ecdsa-sha2-nistp384-cert-v01@openssh.com, ecdsa-sha2-nistp521-cert-v01@openssh.com, ssh-rsa-cert-v01@openssh.com,ssh-dss-cert-v01@openssh.com, ssh-rsa-cert-v00@openssh.com,ssh-dss-cert-v00@openssh.com, ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521, ssh-rsa,ssh-dss** If **hostkeys** are known for the destination host then this default is modified to prefer their algorithms.
HostKeyAlias	Specifies an alias that should be used instead of the real hostname when looking up or saving the host key in the host key database files. This option is useful for tunneling SSH connections or for multiple servers running on a single host.
HostName	Specifies the real hostname to log into. This can be

used to specify nicknames or abbreviations for hosts. If the hostname contains the character sequence '**%h**', then this will be replaced with the hostname specified on the command line (this is useful for manipulating unqualified names). The default is the name given on the command line. Numeric IP addresses are also permitted (both on the command line and in **HostName** specifications).

IdentityFile

Specifies a file from which the user's DSA, ECDSA or DSA authentication identity is read. The default is **~/.ssh/identity** for protocol version 1, and **~/.ssh/id_dsa**, **~/.ssh/id_ecdsa** and **~/.ssh/id_rsa** for protocol version 2. Additionally, any identities represented by the authentication agent will be used for authentication. ssh(1) will try to load certificate information from the file name obtained by appending **-cert.pub** to the path of a specified **IdentityFile**.

The file name may use the tilde syntax to refer to a user's home directory or one of the following escape characters: '**%d**' (local user's home directory), '**%u**' (local user name), '**%l**' (local hostname), '**%h**' (remote hostname) or '**%r**' (remote username).

It is possible to have multiple identity files specified in configuration files; all these identities will be tried in sequence. Multiple **IdentityFile** directives will add to the list of identities tried (this behaviour differs from that of other configuration directives).

IdentitiesOnly

Specifies that **ssh** should only use the authentication identity files configured in the **ssh_config** files, even if **ssh-agent** offers more identities. The argument to this keyword must be "**yes**" or "**no**". This option is intended for situations where **ssh-agent** offers many different identities. The default is "**no**".

IPQoS

Specifies the IPv4 type-of-service or DSCP class for connections. Accepted values are "**af11**", "**af12**", "**af13**", "**af21**", "**af22**", "**af23**", "**af31**", "**af32**", "**af33**", "**af41**", "**af42**", "**af43**", "**cs0**", "**cs1**", "**cs2**", "**cs3**", "**cs4**", "**cs5**", "**cs6**", "**cs7**", "**ef**", "**lowdelay**", "**throughput**", "**reliability**", or a numeric value. This option may take one or two

	arguments, separated by whitespace. If one argument is specified, it is used as the packet class unconditionally. If two values are specified, the first is automatically selected for interactive sessions and the second for non-interactive sessions. The default is "**lowdelay**" for interactive sessions and "throughput" for non-interactive sessions.
KbdInteractiveAuthentication	Specifies whether to use keyboard-interactive authentication. The argument to this keyword must be "**yes**" or "**no**". The default is "**yes**".
KbdInteractiveDevices	Specifies the list of methods to use in keyboard-interactive authentication. Multiple method names must be comma-separated. The default is to use the server specified list. The methods available vary depending on what the server supports. For an **OpenSSH** server, it may be zero or more of: "**bsdauth**", "**pam**", and "**skey**".
KexAlgorithms	Specifies the available KEX (Key Exchange) algorithms. Multiple algorithms must be comma-separated. The default is the following very long string: **ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521, diffie-hellman-group-exchange-sha256, diffie-hellman-group-exchange-sha1, diffie-hellman-group14-sha1, diffie-hellman-group1-sha1**
LogLevel	Gives the verbosity level that is used when logging messages from **ssh**. The possible values are: **QUIET**, **FATAL**, **ERROR**, **INFO**, **VERBOSE**, **DEBUG**, **DEBUG1**, **DEBUG2**, and **DEBUG3**. The default is **INFO**. **DEBUG** and **DEBUG1** are equivalent. **DEBUG2** and **DEBUG3** each specify higher levels of verbose output.
MACs	Specifies the MAC (message authentication code) algorithms in order of preference. The MAC algorithm is used in protocol version 2 for data integrity protection. Multiple algorithms must be comma-separated. The default is the following string: **hmac-md5,hmac-sha1,umac-64@openssh.com, hmac-ripemd160,hmac-sha1-96,hmac-md5-96, hmac-sha2-256,hmac-sha2-256-96,hmac-sha2-**

	512, hmac-sha2-512-96
NoHostAuthenticationForLocalhost	This option can be used if the home directory is shared across machines. In this case localhost will refer to a different machine on each of the machines and the user will get many warnings about changed host keys. However, this option disables host authentication for localhost. The argument to this keyword must be "**yes**" or "**no**". The default is to check the host key for localhost.
NumberOfPasswordPrompts	Specifies the number of password prompts before giving up. The argument to this keyword must be an integer. The default is **3**.
PasswordAuthentication	Specifies whether to use password authentication. The argument to this keyword must be "**yes**" or "**no**". The default is "**yes**".
PKCS11Provider	Specifies which PKCS#11 (Public Key Cryptography Standard number 11) provider to use. The argument to this keyword is the PKCS#11 shared library **ssh** should use to communicate with a PKCS#11 token providing the user's private RSA key.
Port	Specifies the port number to connect on the remote host. The default is **22**.
PreferredAuthentications	Specifies the order in which the client should try protocol 2 authentication methods. This allows a client to prefer one method (e.g., **keyboard-interactive**) over another method (e.g., **password**). The default is: **gssapi-with-mic,hostbased,publickey, keyboard-interactive,password**
Protocol	Specifies the protocol versions **ssh** should support in order of preference. The possible values are '**1**' and '**2**'. Multiple versions must be comma-separated. When this option is set to "**2,1**" **ssh** will try version 2 and fall back to version 1 if version 2 is not available. The default is '**2**'.
ProxyCommand	Specifies the command to use to connect to the server. The command string extends to the end of the line, and is executed with the user's shell. In the command string, any occurrence of '**%h**' will be substituted by the hostname to connect, '**%p**' by the port, and '**%r**' by the remote username. The command can be basically anything, and should read from its standard input and write to its

standard output. It should eventually connect an **sshd** server running on some machine, or execute **sshd -i** somewhere. Host key management will be done using the **HostName** of the host being connected (defaulting to the name typed by the user). Setting the command to "**none**" disables this option entirely. Note that **CheckHostIP** is not available for connects with a proxy command.

This directive is useful in conjunction with **nc** and its proxy support. For example, the following directive would connect via an HTTP proxy at **192.0.2.0**:

ProxyCommand /usr/bin/nc -X connect -x 192.0.2.0:8080 %h %p

PubkeyAuthentication

Specifies whether to try public key authentication. The argument to this keyword must be "**yes**" or "**no**". The default is "**yes**". This option applies to protocol version 2 only.

RekeyLimit

Specifies the maximum amount of data that may be transmitted before the session key is renegotiated. The argument is the number of bytes, with an optional suffix of '**K**', '**M**', or '**G**' to indicate Kilobytes, Megabytes, or Gigabytes, respectively. The default is between '**1G**' and '**4G**', depending on the cipher. This option applies to protocol version 2 only.

RhostsRSAAuthentication

Specifies whether to try rhosts based authentication with RSA host authentication. The argument must be "**yes**" or "**no**". The default is "**no**". This option applies to protocol version 1 only and requires **ssh** to be setuid root.

RSAAuthentication

Specifies whether to try RSA authentication. The argument to this keyword must be "**yes**" or "**no**". RSA authentication will only be attempted if the identity file exists, or an authentication agent is running. The default is "**yes**". Note that this option applies to protocol version 1 only.

SendEnv

Specifies what variables from the local environment should be sent to the server. Note that environment passing is only supported for protocol 2. The server must also support it, and the server must be configured to accept these environment variables. Variables are specified by name, which may contain wildcard characters. Multiple

environment variables may be separated by whitespace or spread across multiple **SendEnv** directives. The default is not to send any environment variables.

ServerAliveInterval

Sets a timeout interval in seconds after which if no data has been received from the server, **ssh** will send a message through the encrypted channel to request a response from the server. The default is **0**, indicating that these messages will not be sent to the server, or **300** if the **BatchMode** option is set. This option applies to protocol version 2 only. **ProtocolKeepAlives** and **SetupTimeOut** are Debian-specific compatibility aliases for this option.

ServerAliveCountMax

Sets the number of server alive messages (see below) which may be sent without **ssh** receiving any messages back from the server. If this threshold is reached while server alive messages are being sent, **ssh** will disconnect from the server, terminating the session. It is important to note that the use of server alive messages is very different from **TCPKeepAlive**. The server alive messages are sent through the encrypted channel and therefore will not be spoofable. The TCP keepalive option enabled by **TCPKeepAlive** is spoofable. The server alive mechanism is valuable when the client or server depend on knowing when a connection has become inactive.

The default value is **3**. If, for example, **ServerAliveInterval** (see below) is set to **15** and **ServerAliveCountMax** is left at the default, if the server becomes unresponsive, **ssh** will disconnect after approximately 45 seconds. This option applies to protocol version 2 only; in protocol version 1 there is no mechanism to request a response from the server to the server alive messages, so disconnection is the responsibility of the TCP stack.

StrictHostKeyChecking

If this flag is set to "**yes**", **ssh** will never automatically add host keys to the **~/.ssh/known_hosts** file, and refuses to connect to hosts whose host key has changed. This provides maximum protection against trojan horse attacks, though it can be annoying when the **/etc/ssh/ssh_known_hosts** file is poorly maintained or when connections to new hosts are

frequently made. This option forces the user to manually add all new hosts. If this flag is set to "**no**", **ssh** will automatically add new host keys to the user known hosts files. If this flag is set to "**ask**", new host keys will be added to the user known host files only after the user has confirmed that is what they really want to do, and **ssh** will refuse to connect to hosts whose host key has changed. The host keys of known hosts will be verified automatically in all cases. The argument must be "**yes**", "**no**", or "**ask**". The default is "**ask**".

TCPKeepAlive

Specifies whether the system should send TCP keepalive messages to the other side. If they are sent, death of the connection or crash of one of the machines will be properly noticed. This option only uses TCP keepalives (as opposed to using **ssh**-level keepalives), so takes a long time to notice when the connection dies. As such, you probably want the **ServerAliveInterval** option as well. However, this means that connections will die if the route is down temporarily, and some people find it annoying.

The default is "**yes**" (to send TCP keepalive messages), and the client will notice if the network goes down or the remote host dies. This is important in scripts, and many users want it too.

To disable TCP keepalive messages, the value should be set to "**no**".

UsePrivilegedPort

Specifies whether to use a privileged port for outgoing connections. The argument must be "**yes**" or "**no**". The default is "**no**". If set to "**yes**", **ssh** must be setuid root. Note that this option must be set to "**yes**" for **RhostsRSAAuthentication** with older servers.

User

Specifies the user to use for log in. This can be useful when a different username is used on different machines. This saves the trouble of having to remember to give the username on the command line.

UserKnownHostsFile

Specifies one or more files to use for the user host key database, separated by whitespace. The default is **~/.ssh/known_hosts**, **~/.ssh/known_hosts2**.

VerifyHostKeyDNS

Specifies whether to verify the remote key using DNS and SSHFP resource records. If this option is

	set to "**yes**", the client will implicitly trust keys that match a secure fingerprint from DNS. Insecure fingerprints will be handled as if this option was set to "**ask**". If this option is set to "**ask**", information on fingerprint match will be displayed, but the user will still need to confirm new host keys according to the StrictHostKeyChecking option. The argument must be "**yes**", "**no**", or "**ask**". The default is "**no**". Note that this option applies to protocol version 2 only.
-P *port*	Specifies the port to connect to on the remote host.
-p	Preserves modification times, access times, and modes from the original files transferred.
-q	Quiet mode: disables the progress meter as well as warning and diagnostic messages from **ssh**.
-R *num_requests*	Specify how many requests may be outstanding at any one time. Increasing this may slightly improve file transfer speed but will increase memory usage. The default is **64** outstanding requests.
-r	Recursively copy entire directories when uploading and downloading. Note that **sftp** does not follow symbolic links encountered in the tree traversal.
-S *program*	Name of the program to use for the encrypted connection. The program must understand **ssh** options.
-s *subsystem* | *sftp_server*	Specifies the **SSH2** subsystem or the path for an **sftp** server on the remote host. A path is useful for using **sftp** over protocol version 1, or when the remote **sshd** does not have an **sftp** subsystem configured.
-v	Raise logging level. This option is also passed to **ssh**.

Interactive mode

In interactive mode, **sftp** logs you into the remote system and places you at a prompt that is similar to the command prompt on your local system. It offers you a limited, but very useful, set of commands with which you can navigate the remote file system and send and receive files.

Let's say you want to start an interactive **sftp** session on the server named "**server.myhost.com**". And, let's say your user account on **server.myhost.com** is named "**user**". From your system's command line, You can start the session using the command:

```
sftp user@server.myhost.com
```

Or, if your username on your *local* system is *also* "**user**", you could type:

```
sftp server.myhost.com
```

...and your username "**user**" will automatically be sent as the username.

The server will respond by asking you for your password:

```
user@server.myhost.com's password:
```

...and if you enter it correctly, you will receive a "you're connected" message, and the **sftp** prompt, like this:

```
Connected to server.myhost.com.
sftp>
```

You can now move around in the filesystem with "**cd** *directory*", list files with "**ls**", download files with "**get** *filename*", and upload files with "**put** *filename*". It's pretty much identical to an interactive **ftp** session, except it's encrypted and secure.

When you're done, you can log off with the "**bye**" command ("**exit**" also works), and **sftp** will exit.

Here's a list of the commands you can use in Interactive Mode:

Interactive mode commands

bye	Close the session and quit **sftp**. Same as "**exit**".
cd *path*	Change the remote working directory to *path*.
chgrp *grp path*	Change group of file *path* to *grp*. *path* may contain wildcard characters and may match multiple files. *grp* must be a numeric GID (group ID).
chmod *mode path*	Change the permissions of file *path* to *mode*. *path* may contain wildcard characters and can match multiple files.
chown *own path*	Change owner of file *path* to *own*. *path* may contain wildcard characters and may match multiple files. *own* must be a numeric UID (user ID).
df [**-hi**] [*path*]	Display usage information for the filesystem holding the current directory (or *path* if specified). If the **-h** flag is specified, the capacity information will be displayed using "human-readable" suffixes. The **-i** flag requests display of inode information in addition to capacity information. This command is only supported on servers that implement the "statvfs@openssh.com" extension.
exit	Close the session and quit **sftp**. Same as "**bye**".
get [**-Ppr**] *remote-path* [*local-path*]	Retrieve the *remote-path* and store it on the local machine. If the local path name is not specified, it is given the same name it has on the remote machine. *remote-path* may contain wildcard characters and may matcal-path is specified, then

local-path must specify a directory.

If either the **-P** or **-p** flag is specified, then full file permissions and access times are copied too.

If the **-r** flag is specified then directories will be copied recursively. Note that **sftp** does not follow symbolic links when performing recursive transfers.

help — Display a help message.

lcd *path* — Change the local working directory to *path*.

lls *[ls-options [path]]* — Display a directory listing of the local directory *path*, or the local working directory if *path* is not specified.

ls-options may contain any flags supported by the local system's **ls** command.

path may contain wildcard characters and may match multiple files.

lmkdir *path* — Create a directory on your local system specified by *path*.

ln [**-s**] *oldpath newpath* — Create a link from *oldpath* to *newpath*. If the **-s** flag is specified the created link is a symbolic link, otherwise it is a hard link.

lpwd — Print the name of the local working directory.

ls [**-1afhlnrSt**] [*path*] — Display a remote directory listing of either *path* or the current directory if *path* is not specified. *path* may contain wildcard characters and can match multiple files.

The following flags alter the behaviour of **ls**:

- **-1** Display the listing in a single column.
- **-a** List files whose names begin with a dot ("**.**"), which otherwise would not be listed.
- **-f** Do not sort the list. The default sort order is lexicographical.
- **-h** When used in conjunction with **-l** or **-n** options (which create a long-format list), this option specifies "human-readable" unit suffixes for file sizes. For instance, instead of "**4096**" bytes, this option will list the size as "**4.0K**".
- **-l** Create a "long-format" list, displaying additional file details including permissions and ownership information.
- **-n** Creates a long-format list Like the **-l** option, but with user and group information presented numerically.
- **-r** Reverse the sort order of the listing.
- **-S** Sort the listing by file size, largest to smallest.
- **-t** Sort the listing by last modification time, newest to oldest.

lumask *umask* — Set local umask to *umask*.

mkdir *path* — Create a directory on the remote system specified by *path*.

progress — Toggle the display of a progress meter.

put [**-Ppr**] — Upload *local-path* and store it on the remote machine. If the remote path name is not

	specified, it is given the same name it has on the local machine. *local-path* may contain wildcard characters and may match multiple files. If it does, and *remote-path* is specified, then *remote-path* must specify a directory.
local-path [*remote-path*]	If either the **-P** or **-p** flag is specified, then full file permissions and access times are copied too. If the **-r** flag is specified then directories will be copied recursively. Note that **sftp** does not follow symbolic links when performing recursive transfers.
pwd	Display the name of the remote working directory.
quit	Quit **sftp**.
rename *oldpath newpath*	Rename a remote file from *oldpath* to *newpath*.
rm *path*	Delete the remote file specified by *path*.
rmdir *path*	Remove the remote directory specified by *path*.
symlink *oldpath newpath*	Create a symbolic link to *oldpath* named *newpath*.
version	Display the **sftp** protocol version.
!*command*	Execute the command *command* in a local shell.
!	Drop into a local shell. Type **exit** to return to the **sftp** session.
?	Same as "**help**".

Notes:

- Interactive Mode commands are case-insensitive, so it doesn't matter if you spell them with capital or lowercase letters (or a mix of both). Filenames are still case-sensitive, however.
- Any file or directory names that contain spaces must be enclosed in quotes, or the server will interpret them as separate names.

Automatic retrieval mode

In this mode, you can specify the exact pathname of the file (or files) you want to retrieve in the **sftp** command itself. For example, if you want to get the file **documents/portfolio.zip** from the remote server **files.myhost.com** (where your username is **myname**), you could use the command:

```
sftp myname@files.myhost.com:documents/portfolio.zip
```

When you run this command, **sftp** will connect to **files.myhost.com**, ask you for your password, and once you're authenticated it will attempt to download the file **documents/portfolio.zip**. Since we didn't put a slash at the beginning of the directory name, it will look for **documents** in your home directory on the server. If it finds **portfolio.zip**, it will download it.

The output will look like this:

```
Fetching /home/myname/documents/portfolio.zip to portfolio.zip
```

...and then **sftp** will exit. You can also specify a location for the file to be downloaded. For instance, this command:

```
sftp myname@files.myhost.com:documents/portfolio.zip /tmp
```

...will download **portfolio.zip** into your **/tmp** directory. Or, you can specify a completely different name for the downloaded file:

```
sftp myname@files.myhost.com:documents/portfolio.zip /tmp/portfolio-new.zip
```

...and the output will indicate the new filename:

```
Fetching /home/myname/documents/portfolio.zip /tmp/porfolio-new.zip
```

You can also specify wildcards in the filename, for instance:

```
sftp myname@files.myhost.com:documents/*.zip
```

...and **sftp** will download any files with the extension **.zip** in the **documents** remote directory. The output will list each file on its own line, like this:

```
Fetching /home/myname/documents/portfolio.zip to portfolio.zip
Fetching /home/myname/documents/resume.zip to resume.zip
Fetching /home/myname/documents/profile-pic.zip to profile-pic.zip
```

Starting interactive mode in a specific remote directory

Sometimes it's more convenient to start an interactive mode session right from a specific remote directory. You can do this by specifying it on the command line:

```
sftp myname@files.myhost.com:documents/budget/april/

myname@files.myhost.com's password:
Connected to files.myhost.com.
Changing to: /home/myname/documents/budget/april/
sftp>
```

Batch mode

It's also possible to run **sftp** in a completely scripted fashion. This is called *batch mode*, and it allows you to perform **sftp** transfers without any interaction at the keyboard. This is useful, for instance, if you want to set up a recurring transfer in a **cron** job, or a one-time scheduled transfer using the **at** command.

However, because batch mode is completely non-interactive, it does not allow you to enter a username and password when connecting to the server. So, to use batch mode, you'll have to log in automatically. The standard way to do this, and the most secure, is to use *public key authentication*. Let's go over that quickly.

Setting Up Public Key Authentication

Public Key Authentication allows you to log into a remote server securely without typing in your password. First, you generate two keys on your local system: a private key and a public key. Then you copy the text of your public key onto the remote server. After that, as long as you have the private key on your local machine, you can log into the remote machine without typing in a password.

To do this, the first step is to generate the public and private keys.

The keys will be located in the directory **.ssh** in your home directory on your local system. First, check to see if the **.ssh** directory already exists:

```
ls -d ~/.ssh
```

This will either return the directory name:

```
/home/username/.ssh
```

...or tell you that it doesn't exist:

```
ls: cannot access /home/username/.ssh: No such file or directory
```

If it doesn't exist, we need to create it before the next step:

```
mkdir ~/.ssh
```

Next, we need to make sure this directory has the correct permissions. You want to ensure that you're the only person who can access this directory (read, write, and execute). For a directory, the octal value of this file mode is **700**. Let's change the permissions on our **.ssh** directory:

```
chmod 700 ~/.ssh
```

Now we need to generate the keys themselves. The program used to generate key pairs for the **ssh** protocol is called **ssh-keygen**. Run it at the command line without any options:

```
ssh-keygen
```

It will prompt you for the information it needs to generate the keys. Use all the default values (press Enter at every prompt).

Note

One of the prompts will ask you for a passphrase, which offers an *additional level* of security on top of the encrypted private key. Here we will leave the password blank. If you want to use a password with your key, you should use a program called **ssh-agent** to load your key into memory; this will allow you to use a password-protected key without having to type in the password more than once.

The output from **ssh-keygen** will look something like this:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/username/.ssh/id_rsa):
Created directory '/home/username/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/username/.ssh/id_rsa.
Your public key has been saved in /home/username/.ssh/id_rsa.pub.
The key fingerprint is:
d4:7b:66:c4:e6:ba:78:87:e7:23:08:c7:0d:d7:b0:7f username@crunch
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|         ...     |
|        . .P+    |
|       .. o=.    |
|     ...S+.e=    |
|      . o .=. E  |
|       o ... .   |
|      ..e.+.+    |
|        ...=..   |
+-----------------+
```

You'll even get a neat piece of art representing your public key, which you can print out and hang on your wall, if you like.

Your keys are now generated. There are two files, **id_rsa** and **id_rsa.pub**. We need to change the permissions on these files as well, so that no one but you can access them (read, write, and execute). The octal value of these permission bits is **700**.

```
chmod 700 ~/.ssh/id_rsa*
```

And make sure the directory has the same permission bits set:

```
chmod 700 ~/.ssh
```

Now **ssh** to your server. Let's say it's called **myhost.com**:

```
ssh myname@myhost.com
```

Enter your password and log in. Once you're at your server's command prompt, check to see if the **.ssh** directory exists there. On the server:

```
ls -d ~/.ssh
```

If it doesn't exist, create it, and give it the appropriate permissions, just like on your local system. On the server:

```
mkdir ~/.ssh
```

```
chmod 700 ~/.ssh
```

Same for the **authorized_keys** file. First check that it exists. On the server:

```
ls ~/.ssh/authorized_keys
```

If it doesn't, create it. You can use **touch** to create an empty file. On the server:

```
touch ~/.ssh/authorized_keys
```

```
chmod 700 ~/.ssh/authorized_keys
```

Of course, if the directory and file exist already, you don't need to create them. Either way, once you know the **~/.ssh/authorized_keys** file exists, you can log out of the server:

```
logout
```

Which will return you to your local system command prompt.

Now you need to place the contents of your local public key file (**~/.ssh/id_rsa.pub**, which you created earlier with **ssh-keygen**) into the file **~/.ssh/authorized_keys** on your server.

The contents of this file are all in one very long line (no line breaks). You can look at it yourself with the **cat** command:

```
cat ~/.ssh/id_rsa.pub
```

...and it will look something like this:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDTiP0LXi74qgpp6VBqzro67QOGtum10
t2epYsOm6kKncf62JVMSlwYH7QwAskxkA6ripvo+TlwRBqqLaF2ACX4CivQkoabqsdFAd
uGcKVICUFZaexUmw2eIEKF4qCOvRDP/Juol1S+ID1glYJRSqDcmAb3jApTRDMXM/w7Tl3
qz5/cp3MINKM3+apBfe7F7iDezjQ/U0HqtH2+Np83u4X2G+LIFnpV0RdalkqCuM6tSv2C
m4FdPazsIwSmFptBKnw00IdIqYpnkQmOJMk47cGDzqczii7KMCy3wRNqkaLwefRB0MZeJ
ipz4+a27kQEqerAIHt37/MMT5XNqn3mqbI myuser@myhostname
```

This line of text needs to be placed into the **authorized_keys** file on your server, on its own line. There are several ways to do this: you could copy the text on your local server, open the file using a text editor on the server, and paste it in on its own line. Or, you could use a program called **ssh-copy-id**, which is part of the default **ssh** installation on many systems. However, here we will append it directly to the file using **ssh** itself.

If your remote username is **myusername** and your server name is **myhost.com**, you would run this command:

```
cat ~/.ssh/id_rsa.pub | ssh myname@myhost.com 'cat >> ~/.ssh/authorized_keys'
```

This runs the **cat** command on your public key file, pipes the output to **ssh**, which takes that input and appends it directly to the **authorized_keys** file on the remote machine.

Now your public key is installed on the server, and you should be able to log in without a password, as well as conduct batch **sftp** sessions.

Note

If the server is still asking you for your password when you try to log in, check that the server's ssh daemon configuration, located by default in **/etc/ssh/sshd_config**, contains the following two lines:

```
RSAAuthentication yes
PubkeyAuthentication yes
```

These are part of the default configuration, so you shouldn't need to add them, set them, or un-comment them in the configuration file. However, they are required for public key authentication. If it's not working, this is the first place you should check.

Executing the batch sftp session

To run a batch **sftp** session, create a text file containing the sequence of sftp commands to be run on the server, with each command on its own line. For instance, if you want to automate the uploading of a set of files called **image01.jpg**, **image02.jpg**... into a directory on the remote server called **images** in your home directory, you could create a text file called **mybatch.txt** which contains the following commands:

```
cd images
put image*.jpg
```

Then, you would execute the batch with the following command:

```
sftp -b mybatch.txt myname@myhost.com
```

...and **sftp** will output the results of the commands, for example:

```
sftp> cd images
sftp> put image*.jpg
Uploading image01.jpg to /home/myname/images/image01.jpg
Uploading image02.jpg to /home/myname/images/image02.jpg
Uploading image03.jpg to /home/myname/images/image03.jpg
```

After all commands have been executed (successfully or not), **sftp** will log out and return you to the command line.

Examples

```
sftp myhost.com
```

This command attempts to initiate an interactive sftp session with the server **myhost.com**. The name used to log in will be the same as the username with which you ran the command. Once you are successfully logged in, you will see a message similar to the following, along with the **sftp>** command prompt:

```
Remote system type is UNIX.
Using ASCII mode to transfer files.
sftp>
```

```
sftp fred@myhost.com
```

Same as the above command, but attempts to log in with the username **fred**.

```
sftp fred@myhost.com:/home/fred/images
```

Attempts to initiate an interactive **sftp** session with the server **myhost.com**, using the name **fred** to log in. Upon successful login, you will begin the session in the directory **/home/fred/images**.

```
sftp fred@myhost.com:/home/fred/images/picture.jpg
```

Attempts to download the file **/home/fred/images/picture.jpg** from the server **myhost.com** using the username **fred** to log in. If the file exists, it will be downloaded to the local working directory, and then **sftp** will exit.

```
sftp -b batch.txt fred@myhost.com
```

Attempts to execute the sftp commands in the text file **batch.txt**, on the server **myhost.com**, as the user named **fred**. The commands in the file **batch.txt** must be listed one per line. For the session to be initiated, the server and local client must be configured so that no keyboard input is required to log in; see Setting Up Public Key Authentication above for more information.

Interactive command examples

The following examples may be run from the **sftp>** prompt once an interactive session has been initiated. See Interactive Mode Commands above for a complete list of interactive commands and options.

```
pwd
```

Prints the name of the remote working directory.

```
lpwd
```

Prints the name of the local working directory.

```
ls
```

List the contents of the remote working directory.

```
lls
```

List the contents of the local working directory.

```
cd documents
```

Changes the remote working directory to the subdirectory **documents**.

```
lcd documents
```

Changes the local working directory to the subdirectory **documents**.

```
get mydocs.zip
```

Download the remote file **mydocs.zip** into the local working directory.

```
get mydocs.zip /home/fred
```

Download the remote file **mydocs.zip** into the local directory **/home/fred**. If the directory **/home/fred** does not exist, **sftp** will attempt to download the file into the local directory **/home** and name it **fred**.

Note

sftp does not recognize the tilde shortcut for home directories ("~"), so you have to use the complete name of a home directory if you're specifying it in **sftp**.

```
get mydocs.zip /home/fred/downloaded-docs.zip
```

Download the remote file **mydocs.zip** into the local directory **/home/fred**, giving it the new name **downloaded-docs.zip** after it is downloaded.

```
mkdir documents
```

Create the directory **documents** in the remote working directory.

```
put documents.zip
```

Upload the local file **documents.zip** into the remote working directory.

```
put /home/fred/documents/documents.zip /home/fred/documents/mydoc.zip
```

Upload the local file **/home/fred/documents/documents.zip** into the remote directory **/home/fred/documents**, renaming it **mydoc.zip** after it is uploaded.

```
put /home/fred/images/image*.jpg /home/fred/images
```

Upload all files in the local directory **/home/fred/images** whose name starts with **image**, and ends in the suffix **.jpg**, into the remote directory **/home/fred/images**.

```
rename /home/fred/file.txt /home/fred/newfile.txt
```

Rename the remote file **/home/fred/file.txt**, giving it the name **newfile.txt**.

```
rm /home/fred/newfile.txt
```

Delete the remote file **/home/fred/newfile.txt**.

```
!commandname option1 option2
```

Run the command *commandname option1 option2* on your local system without disconnecting from the **sftp** session.

```
bye
```

Disconnect from the **sftp** session, and quit **sftp**.

ftp — Conduct an interactive FTP session over a secure network connection.
slogin — Login to a remote system securely.