

# Scripting the OpenSSH, SFTP,



# and SCP Utilities on i

Presented by

Scott Klement

<http://www.scottklement.com>

© 2010-2015, Scott Klement

*Why do programmers get Halloween and Christmas mixed-up?  
31 OCT = 25 DEC*

## Objectives Of This Session



- Setting up OpenSSH on i
- The OpenSSH tools: SSH, SFTP and SCP
- How do you use them?
- How do you automate them so they can be run from native programs (CL programs)

# What is SSH



SSH is short for "Secure Shell."

Created by:

- Tatu Ylönen (SSH Communications Corp)
- Björn Grönvall (OSSH – short lived)
- OpenBSD team (led by Theo de Raadt)

The term "SSH" can refer to a secured network protocol.

It also can refer to the tools that run over that protocol.

- Secure replacement for "telnet"
- Secure replacement for "rcp" (copying files over a network)
- Secure replacement for "ftp"
- Secure replacement for "rexec" (RUNRMTCMD)

3

# What is OpenSSH



"Puffy" – OpenBSD's Mascot

OpenSSH is an open source (free) implementation of SSH.

- Developed by the OpenBSD team
  - but it's available for all major OSes
- Included with many operating systems
  - BSD, Linux, AIX, HP-UX, MacOS X, Novell NetWare, Solaris, Irix... and yes, IBM i.
- Integrated into appliances (routers, switches, etc)
  - HP, Nokia, Cisco, Digi, Dell, Juniper Networks

*The #1 SSH implementation in the world.*

- More than 85% of all SSH installations.
- Measured by ScanSSH software.
- You can be sure your business partners who use SSH will support OpenSSH

4

## Included with IBM i



These must be installed  
(all are free and shipped with IBM i \*\*)

- 57xx-SS1, option 33 = PASE
- 5733-SC1, \*BASE = Portable Utilities
- 5733-SC1, option 1 = OpenSSH, OpenSSL, zlib
- 57xx-SS1, option 30 = QShell (useful, not required)

\*\* in v5r3, had 5733-SC1 had to be ordered separately (no charge.) In v5r4 or later, it's shipped automatically. Starting with v6r1, it's included on the B29xx\_02 CD.

Install these with the CDs/DVDs that came with i

For 5733-SC1:

```
RSTLICPGM LICPGM(5733SC1) DEV(OPTxx)
      OPTION(*BASE) RSTOBJ(*ALL) LNG(2924)

RSTLICPGM LICPGM(5733SC1) DEV(OPTxx)
      OPTION(1) RSTOBJ(*PGM)
```

5

## The PASE Environment



OpenSSH was originally written for a Unix environment. IBM chose to keep the number of changes as small as possible.

- Input/Output in "streams"
- Scrolling command-line (or "shell") interface.
- Hierarchical directory structure (IFS)
- ASCII character set
- Fewer changes means less risk of a mistake that might open up a security hole.

The Portable Application Solutions Environment (PASE) provides Unix compatibility on IBM i.

- Run AIX programs with minimal changes (or no changes)
- Use existing AIX compilers to generate the code
- Provides full Unix environment on i

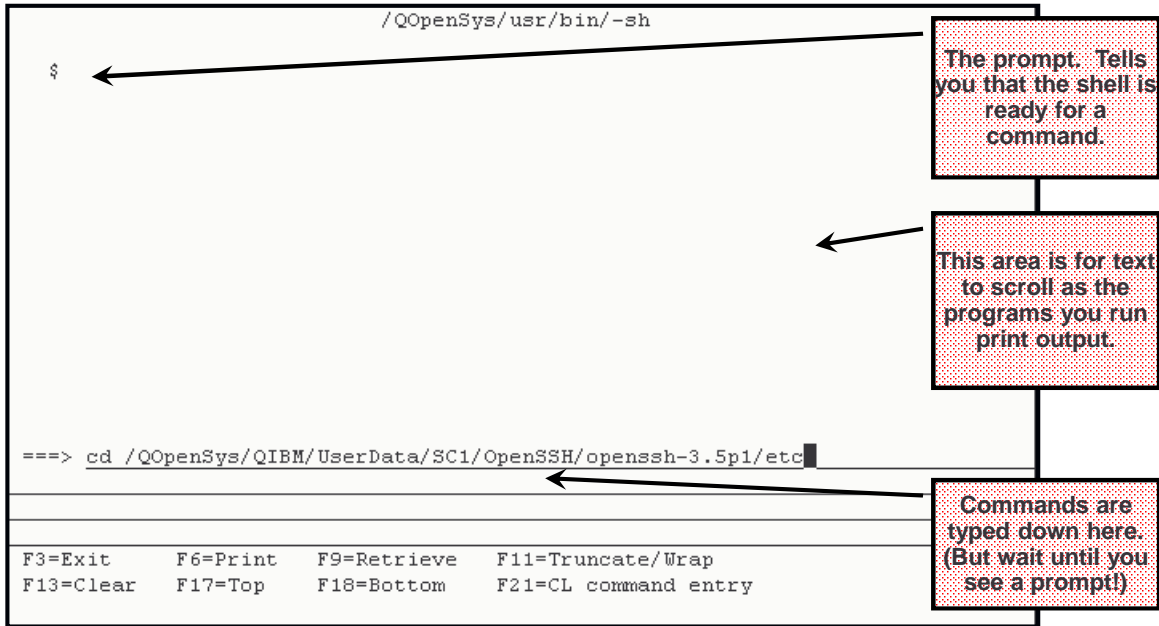
6

# PASE and the Shell



To put yourself at a PASE command line ("shell"), type:

```
CALL QP2TERM
```



# Calling Programs in a Unix Shell



For example, if I typed the following command:

```
cd /tmp
```

- Up to the first space is the program name
- The rest of the line is a series of parameters to be passed to that program, separated by spaces (in this example, there's only one parameter, '/tmp')

Therefore, the preceding Unix command is equivalent to the following syntax at the traditional IBM i command-line:

```
CALL PGM(CD) PARM('/tmp')
```

By contrast, this is calling the program named 'mv', and passing two parameters:

```
mv test.key /home/sklement/.ssh
```

## Quoting Special Shell Characters



There are several characters that have special meanings when typed at a Unix shell.

- Blanks delimit parameters
- Dollar-signs insert variables
- Semi-colons allow more than one command on a line
- Back-slashes mean that the next character is taken literally.
- Ampersands, pipes, greater than, less than all have special meanings
- Characters in quotes do not have special meanings, except:
  - Inside double-quotes ("weak quotes") dollar signs and double quotes
  - Inside single-quotes ("strong quotes") only the single quote itself has a special meaning. Or a backslash if it's followed by a single quote or another backslash.

This example works because the spaces and single quotes do not have special meanings when typed inside weak quotes:

```
cp "Today's Lesson.ppt" "archive/Yesterday's Lesson.ppt"
```

9

## Finding Programs with PATH



To find a program, the PASE shell (like other Unix shells) will search all directories in your PATH environment variable. PATH contains a list of IFS directories to search, separated by colons. Here's an example of setting the PATH from the native environment (prior to calling QP2TERM):

```
ADDENVVAR ENVVAR(PATH) VALUE('/QOpenSys/usr/bin:/dir1:/dir2')
```

If I typed the following command in PASE:

```
mypgm parm1 parm2 parm3
```

PASE would look for a program named 'mypgm' by searching these IFS paths:

- /QOpenSys/usr/bin/mypgm
- /dir1/mypgm
- /dir2/mypgm

Think of PATH the way you think of library lists. (Except it's only used to locate programs -- \*LIBL are used for files and other objects, too.)

10

# Basic PASE Tools



A few commonly used programs (included with PASE):

<code>cd dirname</code>	Change current working directory to <i>dirname</i>
<code>cp src dest</code>	copy a file ( <i>src</i> ) to another name or directory ( <i>dest</i> )
<code>mv old new</code>	Move / rename <i>old</i> file to <i>new</i> name
<code>pwd</code>	Print working directory
<code>mkdir dirname</code>	Create (make) a new directory name <i>dirname</i>
<code>chmod mode file</code>	Change authorities ( <i>mode</i> ) of <i>file</i>
<code>ls</code>	List files (like 'dir' in MS-DOS, or WRKOBJ)
<code>ls -l</code>	List files in long format (more info about the files)
<code>cat file</code>	Dump the contents of <i>file</i> on the screen
<code>rm file</code>	Delete <i>file</i>
<code>find tree expr</code>	Search for files in <i>tree</i> that match <i>expr</i>

More Info about PASE (as well as QShell) is found in the Information Center under Programming -> Shells and Utilities

11

# OpenSSH Tools Provided



<b>ssh</b>	Secure shell client <ul style="list-style-type: none"><li>• Like telnet client (but secure) creates an interactive logon.</li><li>• also works as a 'remote command' tool.</li><li>• can create TCP 'tunnels' that are secured by SSH</li></ul>
<b>scp</b>	Secure copy <ul style="list-style-type: none"><li>• Like Unix cp (copy) command, can copy stream files</li><li>• Copies securely over a network if prefixed by host name</li></ul>
<b>sftp</b>	Secure file transfer program <ul style="list-style-type: none"><li>• Like ftp, but uses the SSH protocol (not FTP protocol) and is secure</li><li>• does not support ASCII/EBCDIC translation.</li><li>• Usually use CPYTOIMPF/CPYFRMIMPF with this tool.</li></ul>
<b>sshd</b>	Secure shell daemon (daemon = server) <ul style="list-style-type: none"><li>• Acts as a server for all ssh tools (ssh, scp, sftp)</li><li>• Interactive logons will be PASE shell logons – allowing true Unix ttys</li><li>• Can be chrooted (user is locked into a given area of the IFS)</li></ul>

Also:

- **ssh-keygen** for generating public and private keys
- **ssh-agent** allows you to load keys into memory for re-use

12

## EXAMPLE: Set up SSHD on IBM i 6.1



For example, let's say you want to be an SSH server.

Acme Foods, Inc. wants to allow customers to order food items by uploading XML files via the 'sftp' or 'scp' tools in SSH.

Assuming that you are tasked with setting up SSHD to allow users to upload (or download) from your server, you'd have to run the following command on IBM i 6.1 (or higher)

```
STRTCPSVR SERVER(*SSHD)
```

That's all. It will automatically generate public/private keys for you the first time it's run. Users can now log on with SSH, SCP or SFTP using their normal IBM i userids and passwords.

13

## EXAMPLE: Set up SSHD on v5r3 or v5r4



(This is not needed on 6.1 or higher -- see previous slide.)

Before you can act as an SSH server, (named *sshd*) you need digital (cryptographic) keys that others can use to verify that you are you.

On i5/OS v5r3 or v5r4:

- Run the following commands from within PASE (CALL QP2TERM):

```
cd /QOpenSys/QIBM/UserData/SC1/OpenSSH/openssh-3.5p1/etc/  
ssh-keygen -N "" -t rsa1 -f ssh_host_key  
ssh-keygen -N "" -t dsa -f ssh_host_dsa_key  
ssh-keygen -N "" -t rsa -f ssh_host_rsa_key
```

Unix commands are case-sensitive. Please match the upper/lower case exactly.

14

## EXAMPLE: Start SSHD on v5r3 or v5r4



(This is not needed on 6.1 or higher -- see previous slides.)

On i5/OS v5r3 or v5r4:

- The easiest way is to use QShell from the native environment:

```
STRQSH CMD(' /QOpenSys/usr/sbin/sshd')
```

- You can also start it from within CALL QP2TERM

```
/QOpenSys/usr/sbin/sshd
```

*Note: Please don't try to start sshd via the QP2SHELL API. Strange results have been noted in that environment. Use QShell (STRQSH) instead.*

15

## It's Started. Now What?



Now that you've started SSHD, users can log on with their IBM i user profiles and passwords.

- Logging on with the SSH tool will give them an interactive PASE command line.
- Logging on with the SCP tool will let them copy files to/from our server.
- Logging on with the SFTP tool will let them copy files to/from the server, plus get directory listings, rename, files, etc. All the things you expect from FTP.

BUT.....

- Just because Acme Foods, Inc is running IBM i doesn't mean their customers are! What if the customers are running Windows? They might try Putty!

16



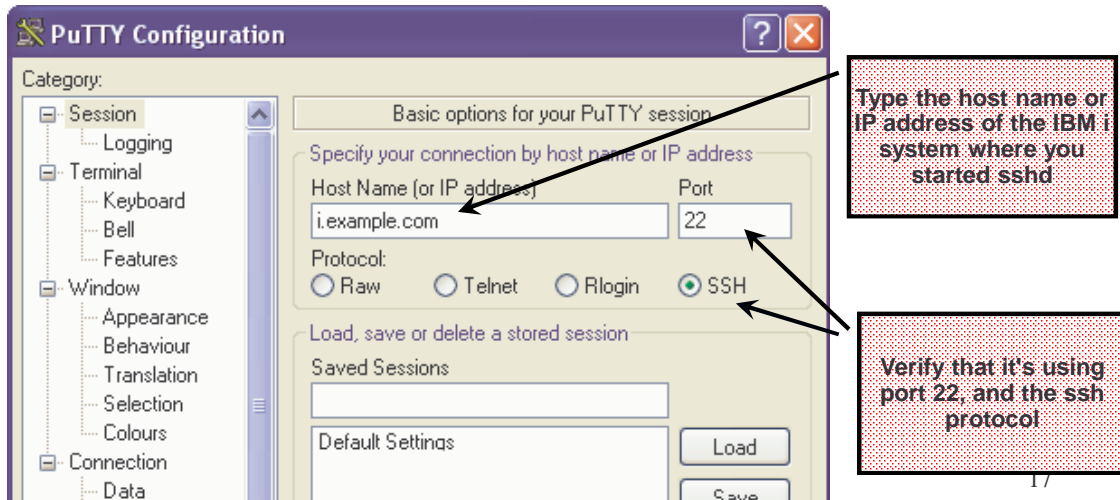
## Test Out SSHD with Putty (1 of 2)



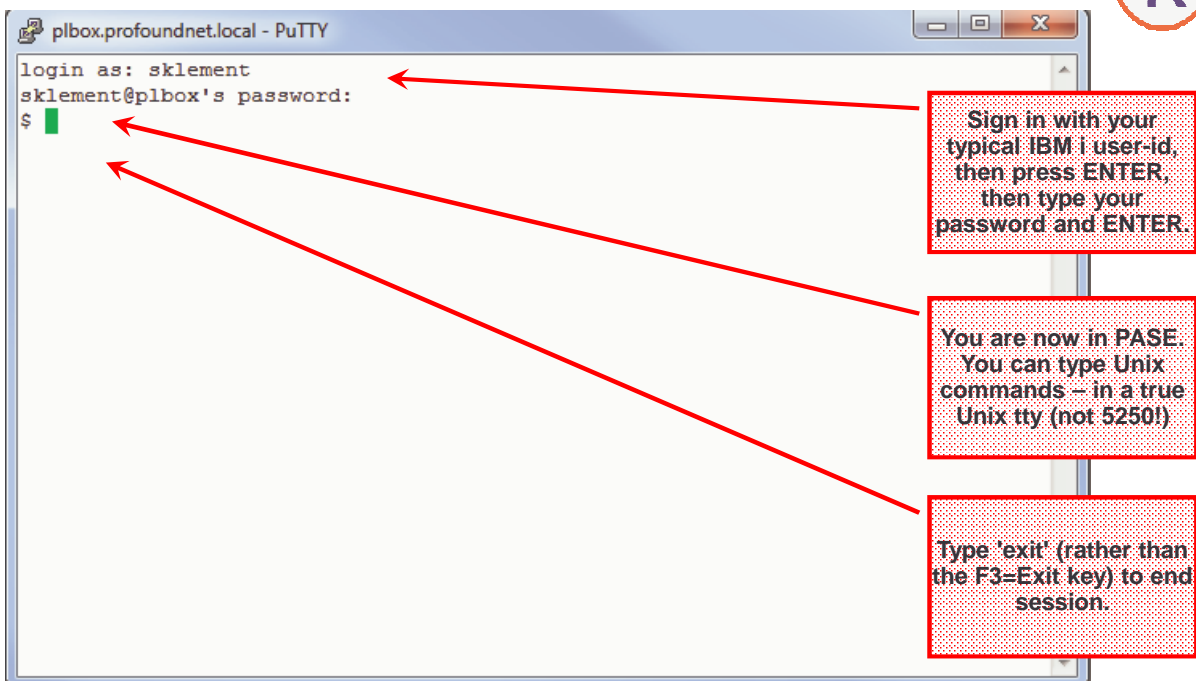
Putty works nicely for Acme Foods' customers who use Windows.

- free ssh software for Windows
- only client-side tools (no server)
- provides putty (ssh), pscp (scp) and psftp (sftp) tools for Windows.

<http://www.chiark.greenend.org.uk/~sgtatham/putty/>



## Test Out SSHD with Putty (2 of 2)



# Test Out SSHD with PSFTP



This is run from a Windows Command Prompt (MS-DOS prompt)

- Start / All Programs / Accessories
- -or- Start / Run / 'cmd'

```

c:\ Command Prompt - psftp i.example.com
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\scotty.HOME>PATH=%PATH%;C:\putty

C:\Documents and Settings\scotty.HOME>psftp i.example.com
login as: klemscot
Using username "klemscot".
klemscot@i.example.com's password:
Remote working directory is /home/KLEMSCOT
psftp>

```

Add the location where Putty resides to your MS-DOS 'PATH'

Sign in with your typical IBM i user-id, then press ENTER, then type your password and ENTER.

You now have a secure FTP session where you can 'get' or 'put' files.

# sshd Server – Closing Thoughts



With the 'sshd' tool running on i, you can:

- Be a server for 'sftp' requests. (Securely transfer files to/from your box.)
- Same with 'scp' (which is often simpler when automating transfers.)
- Server 'ssh' requests – run remote commands securely?
- Interactive logons to PASE.

Why would I want interactive logons to PASE?

- The 5250 terminal is very different from a real Unix terminal (or 'tty')
- Using Putty (or xterm from Linux/Unix) and ssh gives a true Unix terminal
- Useful for programs that are strict about terminal I/O.

Common 'gotchas'

- Due to a limitation in AIX (not in ssh) userids need to be 8 chars or shorter
- sshd must be started by a profile with \*ALLOBJ authority.
- All ssh services run on port 22. This must be open through firewalls.
- LMTCPB(\*YES) has no impact on sshd, but object-level authority works.
- Perhaps allows too much access to your system?
- Restrict access with chroot (*see link at end of presentation*)

# Client Side Tools – Initial Setup



SSH will store certain files in the `.ssh` subdirectory of your home directory

- Home directory is defined in user profile (CHGUSRPRF / CHGPRF)
- By default, it's `/home/my-user-id`  
From PASE: `mkdir /home/my-user-id`
- So SSH files go in `/home/my-user-id/.ssh`  
From PASE: `mkdir /home/my-user-id/.ssh`
- Home dir cannot allow public write access (would open security hole)  
From PASE: `chmod go-w /home/my-user-id`
- `.ssh` directory cannot allow public access at all (same reason)  
From PASE: `chmod go-rwx /home/my-user-id/.ssh`

**Remember:** Adopted authority doesn't work in the IFS.

- Home directory will be based on the real user's home directory.
- Authority to files is based on the real user's authority, not adopted user's.
- If you really want to, you can swap userids with APIs.  
`QSYGETPH, QWTSETP, QSYRLSPH`
- These APIs are also useful to working around the 8-char userid problem.

21

# Client-side SSH Tool



The 'ssh' command in PASE gives you an interactive logon to another computer (like the 'putty' command did in Windows)

```
tn5250 - ssl:as400.klements.com
File Edit View Help
/Q0penSys/usr/bin/-sh

$
> ssh -l klemscot bcserv2.klements.com
The authenticity of host 'bcserv2.klements.com (192.168.5.165)' can't be established.
. key fingerprint is DSA.
Are you sure you want to continue connecting (yes/no)?
```

SSH checks to see if host is in your known\_hosts file. Once you say 'yes', it remembers the digital key from that host. It verifies that it's always the same.

The digital key is saved in the known\_hosts file in your .ssh directory.

F3=Exit F6=Print F9=Retrieve F11=Truncate/Wrap  
F13=Clear F17=Top F18=Bottom F21=CL command entry

5250 007/018

22

## Passwords vs. Digital Keys



*SSH! Don't give away your password to whomever might be listening!*

23

## Public Key Cryptography



When the server was established, a public/private key pair was generated.

- data encrypted with public key requires private key to decrypt
- public keys can be freely shared.
- private keys are never shared.

Server sent us their public key.

- ssh saved it to a file.
- future connects verify that the key is the same
- data is encrypted with the public key and sent to the server. If the server can decrypt it, it proves who they are (nobody else has the private key, so nobody else could decrypt our data.)
- man-in-the-middle attacks fail because they won't have the private key.

*So we know we're talking to the right server, and only it can read our data!*

But .. So far, we haven't given the server our public key. Instead, we've authenticated with a user-id and password.

- not nearly as secure as cryptographic keys – but is still very common.

24

# Digital Keys



Studies have shown passwords to be one of the weak links in security.

- A good password is long and random (and impossible to remember!)
- Most passwords are 8-15 characters long. (Easy to crack.)
- Subject to social-engineering attacks
- Subject to phishing attacks, man-in-the-middle attacks
- When coded into a script, a password is visible to anyone with access to source code or the ability to dump or debug the object.

Studies have shown passwords to be one of the weak links in security.

- Bruce Schneier noted in 2006 that 55% of passwords on MySpace would be crackable in 8 hours with commercially available software.
- CERN analyzing an attack in 1998, it was found that the attacker (with help of software) had successfully guessed more than 47,000 passwords on a system with 186,000 accounts. This was done by taking common passwords from other sites.

*Digital keys provide long, random, cryptographically verifiable "passwords" (authentication strings) that the user doesn't have to remember.*

25

# Establishing Digital Keys



SSH supports three types of keys:

- rsa1 = RSA key for protocol version 1.
- rsa = RSA key for protocol version 2 (default & most secure)
- dsa = DSA key for protocol version 2.

To generate a key for client-side use (shared by ssh, scp and sftp)

- Log on as the user who will be running the ssh, scp or sftp client.
- Type: `CALL PGM(QP2TERM)`
- Type: `ssh-keygen -t rsa -N ""`
- Press ENTER to accept default dir (`/home/userid/.ssh/id_rsa`)

In the directory (above):

- Private key is now stored in the `id_rsa` file.
- Public key is now stored in the `id_rsa.pub` file

This stuff is always done on the "client side" (the side running the 'sftp', 'scp', or 'ssh' program - not the sshd server.)

*Never give the `id_rsa` file to anyone. Protect it with object-level security.*

*The `id_rsa.pub` file should be sent to the server-side to enable logins.*

26

## Installing a Digital Key on the Server



To allow a public key to be used in place of a password:

- Transfer `id_rsa.pub` to the server.
- Add the contents of `id_rsa.pub` to the end of the `authorized_keys` file.

Ways to transfer the `id_rsa.pub` file:

- Use traditional FTP.
- Use the ssh tools (`ssh`, `scp` or `sftp`) with password authentication
- Use Windows Networking (/QNTC or NetServer)
- Use iNav to get the key to your PC, then transfer in E-mail or similar.

The best solution will depend on whomever administers the ssh server. For servers managed by 3rd-parties, you'll usually want to download the public key to your PC, and e-mail it to the administrator.

If you are running the server, and someone has sent you a key, add it to the that user's authorized keys file, from within PASE type:

```
cat /tmp/id_rsa.pub >> /home/user-id/.ssh/authorized_keys
```

27

## Back to the SSH Tool



The 'ssh' command in PASE gives you an interactive logon to another computer (like the 'putty' command did in Windows)

For an interactive logon:

```
ssh -l remote-user-id host.example.com
```

- without `-l`, assumes remote user name is same as local one.

To run a remote command (without interactive logon):

```
ssh -n -l scottk mybank.example.com command-to-run
```

- `-n` disables input to the remote command (required in batch)
- `-l scottk` is the userid I want to log in with.
- `command-to-run` is a command to run on the remote host.  
for `sshd` on IBM i, this is a PASE command  
to run a native command, you can use the 'system' tool.

28

# Running Remote Commands



Log on to Unix machine, switch directory, list directory to an IFS file

```
ssh -l scottk unix.example.com cd /tmp "&&" ls -l > dirlist.txt
```

Log on to Unix machine, run "process daily" script.

```
ssh -l scottk unix.example.com /usr/bin/process_daily.sh
```

Run a native command on an IBM i server – maybe executed from a Unix server? (*this would be entered as one long command. line wrapping added to make slide easier to read*)

```
ssh -l sklement i.example.com
      system \"sndmsg msg('Processing complete. Have a nice
              day!') tousr(sklement)\"
```

29

# Passwords Don't Work

(*but see the later slides that make them work, anyway*)



Another reason to use digital keys:

- passwords do work with ssh tool on a 5250 terminal
- passwords don't work with scp on 5250
- passwords don't work with sftp on 5250

```
> sftp mybank.example.com
Connecting to mybank.example.com...
Host key verification failed.
Connection closed
$
```

*This isn't true of a "real" Unix terminal, however. If you set up sshd and connect with Putty, you can use passwords with scp or sftp.*

```
$ sftp mybank.example.com
Connecting to mybank.example.com...
Password:
sftp>
```

30



# The Secure Copy Tool



```
scp user@host:from-file user@host:to-file
```

- copy (duplicate) the from-file to the to-file
- user@ is optional. if not given, the local user name is assumed.
- host: is optional. if not given, the local host is assumed.

```
scp scottk@mybank.example.com:/home/klement/daily.dat  
/custs/daily/daily.csv
```

- Logs on to “my bank” (mybank.example.com using ssh protocol.)
- signs in with user = scottk
- copies from bank’s file named /home/klement/daily.dat
- Copied to local file /custs/daily/daily.csv

```
scp /custs/daily/daily.csv randi@simple.com:/var/uploads/daily.dat
```

- other direction... copies local file to remote system.

31

## Automating (Scripting) the SCP Tool



```
PGM  
DCL VAR(&RMTFILE) TYPE(*CHAR) LEN(100) +  
  VALUE('check_reconciliation.txt')  
DCL VAR(&LCLFILE) TYPE(*CHAR) LEN(100) +  
  VALUE('/usr/secure/checks.txt')  
DCL VAR(&CMD) TYPE(*CHAR) LEN(500) +  
  
CHGVAR VAR(&CMD) VALUE('PATH=$PATH:/QOpenSys/usr/bin && +  
  scp scottk@mybank.example.com:'' +  
  *CAT &RMTFILE +  
  *TCAT '' '' +  
  *CAT &LCLFILE +  
  *TCAT ''')  
  
ADDENVVAR ENVVAR(QIBM_QSH_CMD_OUTPUT) +  
  VALUE(NONE) +  
  REPLACE(*YES)  
  
ADDENVVAR ENVVAR(QIBM_QSH_CMD_ESCAPE_MSG) VALUE(Y) +  
  REPLACE(*YES)  
  
QSH CMD(&CMD)  
MONMSG MSGID(QSH0000) EXEC(DO)  
  SNDMSG MSG('File transfer failed!') +  
  TOUSR(SKLEMENT)  
  
ENDDO  
  
ENDPGM
```

SCP is much easier for automated file transfers than SFTP, because the whole process can be done in one line.

QIBM.QSH.CMD.OUTPUT controls whether any messages are printed on the screen (or not)

QIBM.QSH.CMD.ESCAPE.MSG causes an \*ESCAPE message to be sent when a file transfer fails, MONMSG is used to capture that escape message.



# The Secure File Transfer Program (SFTP)



```
sftp scottk@mybank.example.com
```

- Logs on to mybank.example.com using ssh protocol.
- signs in with user = scottk
- puts you at a command-prompt
- you can use `get`, `put`, `cd`, `lcd`, `rm`, `rmdir`, `rename` commands

```
sftp -b batch-script scottk@mybank.example.com
```

- Logs on to mybank.example.com using ssh protocol.
- signs in with user = scottk
- commands to run on host are read from a (local) IFS file named `batch-script`
- Script must be in ASCII, each line terminated with LF (in Unix style)

33

## Automating the SFTP Tool



To create a batch script for SFTP, the script must be an ASCII file. You can create one in the (local) IFS as follows:

```
STRQSH CMD('touch -C 819 /tmp/myscript.ftps')
```

Now use the EDTF command to edit the script. (Use F15 and set EOL to \*LF) type the following:

```
cd /edi/outgoing/klement
get file1.edi
rm file1.edi
get file2.edi
rm file2.edi
```

Now you can run the script as follows:

```
sftp -b /tmp/myscript.sftp scottk@mybank.example.com
```

34

# Error Handling in SFTP



SFTP will stop running when:

- it reaches the end of the script, and no errors occur (success!)
- one of the commands fails (failure!)
- unlike traditional ftp, sftp does not normally continue if a command fails.
- if you'd like it to ignore an error, you may precede a command with -
- due to a bug, this "stop on error" didn't work in the early versions of 5733SC1, but IBM fixed it with a PTF. (V5R3 SI25208 / V5R4 SI25209)

```
cd /edi/incoming/klement
-rm file1.edi
put file1.edi
-rm file2.edi
put file2.edi
```

In this example:

- if file1 exists, I want to delete it. But if not, I don't want the script to stop.

35

# Automating SFTP with CL



```
PGM
DCL VAR(&USER) TYPE(*CHAR) LEN(10) +
  VALUE('scottk')
DCL VAR(&HOST) TYPE(*CHAR) LEN(100) +
  VALUE('mybank.example.com')
DCL VAR(&CMD) TYPE(*CHAR) LEN(500)

ADDENVVAR ENVVAR(SFTP_USER) VALUE(&USER) REPLACE(*YES)
ADDENVVAR ENVVAR(SFTP_HOST) VALUE(&HOST) REPLACE(*YES)

CHGVAR VAR(&CMD) VALUE('PATH=$PATH:/QOpenSys/usr/bin && +
  sftp -b /tmp/myscript.sftp $SFTP_USER@$SFTP_HOST')

ADDENVVAR ENVVAR(QIBM_QSH_CMD_OUTPUT) +
  VALUE('FILEAPPEND=/tmp/sftplog.txt') +
  REPLACE(*YES)

ADDENVVAR ENVVAR(QIBM_QSH_CMD_ESCAPE_MSG) VALUE(Y) +
  REPLACE(*YES)

QSH CMD(&CMD)
MONMSG MSGID(QSH0000) EXEC(DO)
  SNDMSG MSG('File transfer failed! See /tmp/sftplog.txt') +
  TOUSR(SKLEMENT)
ENDDO
ENDPGM
```

You can set any ENVVAR you like with the ADDENVVAR command.

ENVVARs can be inserted into a Unix command-line by preceding the variable name with \$

FILEAPPEND= allows output to be appended to an IFS file.

30

## Passwords Do Work



Despite my earlier statement (as well as statements made in some IBM documents!) password authentication can be made to work with SCP and SFTP.

They will work as long as the password seems to be typed from a valid Unix terminal (which means they can't come from a 5250 terminal or be supplied by a CL program or SFTP script.)

Furthermore, passwords aren't a good idea.

- but what if you had an important cust who required password auth?
- The customer is always right!
- You don't really have a choice.

There's a tool called *Expect* that's designed to automate any Unix tool.

37

## The *Expect* Tool



Expect is free software, developed by Don Libes at the U.S. National Institute of Standards and Technology (NIST).

*(See download link at the end of this presentation)*

- Looks (to SSH & friends) like a true Unix terminal
- Has the ability to "type" for the user (send strings)
- Has the ability to scan output from program (SFTP in this example) for key phrases, and act upon them
- Expect isn't purely for SFTP and SCP. It can be used with any Unix utility.

Since it acts like a real terminal, Expect can be used to send a password to any of the OpenSSH tools.

Expecting has a sophisticated scripting language, designed for automating tasks on Unix machines.

38

# Sample Expect Script



```
#!/usr/local/bin/expect -f
set timeout 20
spawn sftp $env(SSH_USER)@$env(SSH_HOST)
expect {
  default {exit 2}
  "continue connecting (yes/no)?" {send "yes\n"; exp_continue}
  "assword:" {send "$env(SSH_PASS)\n"; exp_continue}
  "sftp>"
}
send "put direct_deposit.ach\n"
expect {
  default {exit 2}
  "not found" {exit 3}
  "sftp>"
}
send "quit\n"
exit 0
```

Any where you see \$env(SOMETHING) it represents an environment variable.

This makes it easy to pass data from CL programs, just set a variable with ADDENVVAR.

When expect exits with 0, it will look like success to the CL program.

When it exits with non-zero, it will look like an error.

39

# Running Expect from a CL Program



```
DCL VAR(&USER) TYPE(*CHAR) LEN(15) VALUE('scottk')
DCL VAR(&PASS) TYPE(*CHAR) LEN(15) VALUE('bigboy')
DCL VAR(&HOST) TYPE(*CHAR) LEN(50) +
  VALUE('mybank.example.com')
DCL VAR(&CMD) TYPE(*CHAR) LEN(500) +

CHGVAR VAR(&CMD) VALUE('PATH=$PATH:/QOpenSys/usr/bin+
                       :/QOpenSys/usr/local/bin && +
                       expect -f myscript.exp')

ADDENVVAR ENVVAR(SSH_USER) VALUE(&USER) REPLACE(*YES)
ADDENVVAR ENVVAR(SSH_HOST) VALUE(&HOST) REPLACE(*YES)
ADDENVVAR ENVVAR(SSH_PASS) VALUE(&PASS) REPLACE(*YES)

ADDENVVAR ENVVAR(QIBM_QSH_CMD_OUTPUT) +
  VALUE('FILE=/tmp/expect.log') +
  REPLACE(*YES)

ADDENVVAR ENVVAR(QIBM_QSH_CMD_ESCAPE_MSG) VALUE(Y) +
  REPLACE(*YES)

QSH CMD(&CMD)
MONMSG MSGID(QSH0000) EXEC(DO)
  SNDMSG MSG('File transfer failed! See /tmp/expect.log') +
  TOUSR(SKLEMENT)

ENDDO
```

Notice the ADDENVVARs match the \$env(VAR) in Expect.

FILE=/tmp/expect.log  
the 'FILE' part causes the file to be replaced each time.

40

# More Information



To learn more about using OpenSSH on IBM i:

IBM Porting Central (Official site of 5733-SC1)

<http://www-03.ibm.com/servers/enable/site/porting/tools/openssh.html>

IBM Technote: Using chroot to Restrict ssh to specific directories

<http://www-01.ibm.com/support/docview.wss?uid=nas1eafce9e44f206348862575ce007c7619>

OpenSSH information on ScottKlement.com

- Links to all of Scott's articles about SSH
- Manual pages for OpenSSH tools
- Additional links to IBM resources

<http://www.scottklement.com/openssh/>

Expect (official site):

<http://expect.nist.gov>

Expect download for PASE:

<http://www.scottklement.com/expect/>

41

# This Presentation



You can download a PDF copy of this presentation from:

<http://www.scottklement.com/presentations/>

# Thank you!

42