

Accessing remote Unix systems

Bob Dowling

rjd4@cam.ac.uk

20 July 2006

Table of Contents

| | |
|---|----|
| Accessing remote Unix systems | 1 |
| Introduction..... | 2 |
| Notes..... | 2 |
| Attacks..... | 3 |
| Snooping..... | 3 |
| Fake identity..... | 3 |
| Man in the middle..... | 4 |
| Logging on to a remote system..... | 5 |
| The basic command..... | 5 |
| The first time..... | 5 |
| The password..... | 5 |
| The welcome message..... | 6 |
| The remote session..... | 6 |
| Disconnection..... | 6 |
| Reconnection..... | 6 |
| Exercise..... | 6 |
| First time access and fingerprints..... | 7 |
| Exercise..... | 8 |
| When it goes wrong..... | 8 |
| Transferring files..... | 10 |
| Exercise..... | 10 |
| Using scp..... | 10 |
| Exercise..... | 11 |
| Using sftp..... | 11 |
| The rsync command..... | 14 |
| Exercise..... | 18 |
| Doing without the password..... | 19 |
| Creating the RSA keys..... | 19 |
| Transferring the public key..... | 19 |
| Exercise..... | 20 |
| Appendix..... | 21 |
| SSH fingerprints..... | 21 |
| Setting your prompt..... | 21 |

Introduction

This course gives a quick overview of using the SSH utilities to ensure easy and secure access to remote Unix systems. The SSH server (required on the Unix system you are connecting to) is essentially universal these days and servers supporting version 2 of the protocol can be reasonably demanded of any system administrator. Similarly the SSH clients are now a standard part of all Unix and Linux releases (including all modern Linux distributions, Solaris 2, MacOS X, etc.) and are supported on Windows through the PuTTY program.

The course is designed to be given on PWF Linux workstations, but the principles should apply to almost all Unix and Linux systems that aren't deliberately perversely configured.

Once upon a time the Internet was small, having so few systems on it that every one of them could be listed in a single text file (called `/etc/hosts` on Unix systems) and everyone could be trusted. In this environment a programs were developed for remotely logging on (`telnet`) and for file transfer (`ftp`). For the pure unix-to-unix world a set of even easier programs were written called `rlogin`, `rsh` and `rcp` ("remote login", "remote shell" and "remote copy"). These became know as the "r*" ("r-star") programs.

Today there are hundreds of millions of Internet-connected computers and nobody can be trusted. In particular, nobody can be trusted not to pretend to be a machine they are not or not to listen in on the network to traffic that's not meant for them. If you pass your userid and password over the Internet in plain text¹ they will get snooped eventually. This is a sad, but accurate reflection of the contemporary computing world. Using `telnet`, `ftp` or any of the `r*` commands is a sure-fire way to compromise your account.

To deal with this problem, and a few others too, we provide the SSH suite of programs, which might be regarded as the "s*" programs:

- "secure login": `slogin`
This program is used to log in to remote systems and to provide an interactive shell running at the far end.
- "secure shell": `ssh`
This program is used to run a (typically single) command on a remote system.
- "secure copy": `scp`
This program is used to transfer files to and from other systems. It is joined with `sftp` and `rsync` (over a secure channel) to provide a suite of programs for transferring files in various different ways according to circumstance.

Notes

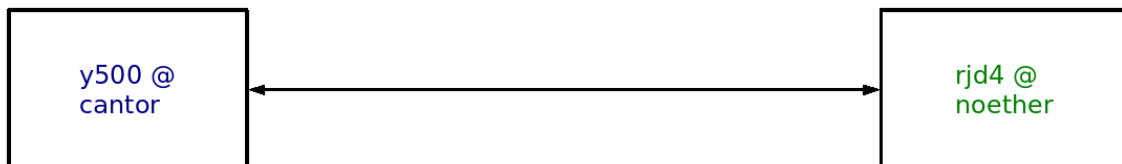
1. The examples here were all run on a Linux workstation but should work equally well from any Unix workstation or other Unix system.
2. The local system is called `cantor` with an account `y500` and the remote system is called `noether` on which my account is called `rjd4`. On both systems the prompt shows the machine and the user. Activity on the local system will be shown in **blue** and on the remote system in **green**.
3. To benefit most from this course you will have an account on a remote Unix system. You can use the PWF Linux servers for everything except the passwordless access chapter. A conventional Unix system should manages for everything.
4. Please also note that this is not a course on cryptography or number theory. The University's Computer Laboratory and Pure Maths Department run perfectly good lecture courses to cover this aspect of the system

¹ "Plain text" is any unencrypted data, as opposed to "cypher text" which is encrypted.

Attacks

So how can your communications be attacked? Well, there are very many ways, but the most common are quickly described here. Consider these cases where user y500 on machine cantor is trying to log in as user rjd4 on machine noether.

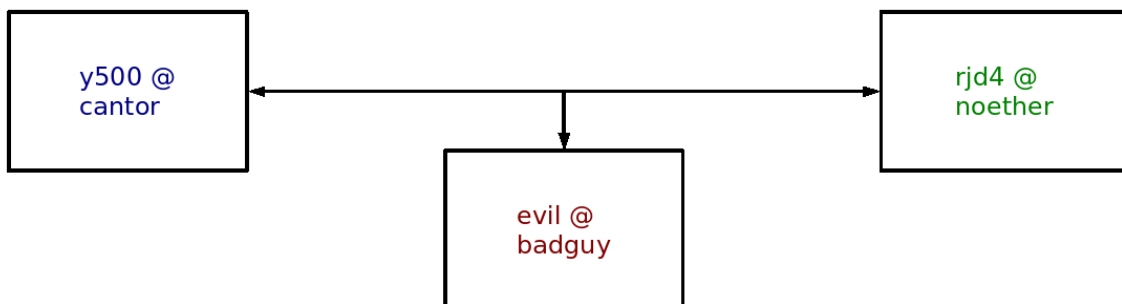
The situation is meant to look like this:



but instead the following might happen.

Snooping

Snooping is the “passive attack”. The bad guy simply gets a copy of all the traffic on the wire between cantor and noether.



The purpose of encryption is that the bad guy can't understand any of what he gets.

Fake identity

The “fake identity” attack is where the bad guy pretends to be the machine noether when y500 attempts the connection from cantor.



A common version of this involves the bad guy suppressing noether (by distributed denial of service attacks, unplugging the wire from the wall or other, more sophisticated means) and letting the connection be made. The bad guy faithfully records the userid and password used,

Accessing remote Unix systems

then prints a spurious error message and breaks the connection with cantor. The next time y500 on cantor tries to make the connection, the suppression of noether is dropped and y500 connects perfectly happily. The bad guy however has a copy of rjd4's password on noether.

Man in the middle

The more insidious version of the previous trick is for the bad guy to pretend to be noether when talking to cantor and simultaneously to pretend to be cantor when speaking to noether.



Both systems set up encrypted connections with the bad guy getting the unencrypted data which he then re-encrypts and passes on.

Logging on to a remote system

So let's run a simple example. We are going to connect from the y500 account on a machine cantor.csi.cam.ac.uk to the rjd4 account on a machine noether.csi.cam.ac.uk with the slogin command. We colour code text according to the machine it comes from.

```
y500@cantor$ slogin rjd4@noether.csi.cam.ac.uk
The authenticity of host 'noether.csi.cam.ac.uk (131.111.10.87)' can't be
established.
RSA key fingerprint is 06:d0:42:b2:ea:74:41:c5:49:80:9d:7b:31:44:00:47.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'noether.csi.cam.ac.uk,131.111.10.87' (RSA) to the
list of known hosts.
Password: not shown
Last login: Wed Mar 29 12:12:42 2006 from euclid.csi.cam.,ac.uk
Have a lot of fun...
rjd4@noether$
rjd4@noether$ uname -n
noether
rjd4@noether$
rjd4@noether$ logout
Connection to noether.csi.cam.ac.uk closed.
y500@cantor$
```

We will examine this a block at a time.

The basic command

```
y500@cantor$ slogin rjd4@noether.csi.cam.ac.uk
```

The slogin command takes one principal argument which combines the account on the remote system and the name of the remote system. The argument “rjd4@noether.csi.cam.ac.uk” means “the rjd4 account on the system noether.csi.cam.ac.uk”. If the “rjd4@” bit is omitted then the connection is to an account at the far end with the same name as the account at the local end. So in this example it would be y500 on noether.

The first time

```
The authenticity of host 'noether.csi.cam.ac.uk (131.111.10.87)' can't be
established.
RSA key fingerprint is 06:d0:42:b2:ea:74:41:c5:49:80:9d:7b:31:44:00:47.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'noether.csi.cam.ac.uk,131.111.10.87' (RSA) to the
list of known hosts.
```

The first time (and only the first time) you connect to a particular system there is an extra challenge. We will examine this in detail in the next section. For now, we will just type “yes” at the prompt.

The password

```
Password: not shown
```

We are challenged for our password just to prove that we are entitled to use the account we are trying to connect to. Obviously, the password is not repeated on the screen. You typically get three attempts to get it right.

The welcome message

```
Last login: Wed Mar 29 12:12:42 2006 from euclid.csi.cam.,ac.uk
Have a lot of fun...
```

The next lines we see come from the remote system and are typically the welcoming message, or “message of the day” as it is on many systems. On a standard SuSE Linux system such as noether this consists of a message identifying the last login by this account and the contents of the file /etc/motd (**m**essage **o**f **t**he **d**ay) which is “Have a lot of fun...” in this case.

The remote session

```
rjd4@noether$
rjd4@noether$ uname -n
noether
rjd4@noether$
rjd4@noether$ logout
```

Then the remote system gives us a prompt (“rjd4@noether\$” in this case) and we can start issuing commands. In this example we only give two commands. The “uname -n” command asks the system to identify itself and “logout” ends the session.

Disconnection

```
Connection to noether.csi.cam.ac.uk closed.
y500@cantor$
```

When the remote session ends we get a confirmation from the local `slogin` program that it has disconnected and finished and then we are back to our local session to carry on our local work.

Reconnection

The next time we connect from cantor to noether we do not get the fingerprint challenge but just the usual password prompt:

```
y500@cantor$ slogin rjd4@noether.csi.cam.ac.uk
Password: not shown
Last login: Mon Apr  3 14:57:37 2006 from cantor.csi.cam.ac.uk
Have a lot of fun...
noether:~$ echo 'Back again!'
Back again!
noether:~$ logout
Connection to noether.csi.cam.ac.uk closed.
y500@cantor$
```

Exercise

If you have an account on the PWF (be it either a course temporary account or your usual permanent account) then you can use it to connect to the PWF Linux servers. Make an `slogin` connection to `linux.pwf.cam.ac.uk` using your PWF account ID and password. You may safely ignore the message about IP addresses; if there was something wrong you would not be allowed to pass. You won't get the full fingerprint challenge because the fingerprints for these hosts are stored centrally.

First time access and fingerprints

Now we need to address that fingerprint challenge we saw the first time we connected.

```
y500@cantor$ slogin rjd4@noether.csi.cam.ac.uk
The authenticity of host 'noether.csi.cam.ac.uk (131.111.10.87)' can't be
established.
RSA key fingerprint is 06:d0:42:b2:ea:74:41:c5:49:80:9d:7b:31:44:00:47.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'noether.csi.cam.ac.uk,131.111.10.87' (RSA) to the
list of known hosts.
Password: not shown
Last login: Wed Mar 29 12:12:42 2006 from euclid.csi.cam.,ac.uk
Have a lot of fun...
rjd4@noether$
rjd4@noether$ uname -n
noether
rjd4@noether$
rjd4@noether$ logout
Connection to noether.csi.cam.ac.uk closed.
y500@cantor$
```

The machine cantor makes a connection via the Internet designed to have the machine noether on the other end. But we may end up connected to the bad guy's machine. This is where the fingerprint comes in.

Each machine running SSH services has a piece of data, visible only to the administration account, called its "secret key". It is important that this piece of data be kept secret. On a standard installation that data is kept in a file `/etc/ssh/ssh_host_rsa_key` and looks like this:

```
-----BEGIN RSA PRIVATE KEY-----
MIICWwIBAAKBgQC9pQ6aE14616kGkfM4jPBA86Cfa80/r9papa0q3ryWjH++mDRk
AjazUz8rhrHsUWXSZcQa10VWrRgjrqkD06TFb6px/8dlc3k5HHEK2jG0yzZSfsKK
...
05cId15+GD3WT04UBwJAC5f3kZzmAbitj05DErdsNyZG4sjpbD7QsfUBeks4DU2F
s2VglxchuiYKzIi6r/DzwBhX5MNlWl1kDMemeEge+w==
-----END RSA PRIVATE KEY-----
```

While that private key must be kept secret, certain complementary bits of information about it are allowed out. One of these is its "fingerprint". The fingerprint of that private key is a list of numbers presented as `06:d0:42:b2:ea:74:41:c5:49:80:9d:7b:31:44:00:47`. There is a function which generates a fingerprint from a secret key but there is no function to get the secret key back from the fingerprint.

Through the magic of cryptography, cantor can get at the fingerprint of the private key of the machine it has connected to in a secure fashion. If the bad guy does not know noether's secret key then he cannot fake this exchange with cantor. So, on initial connection with the machine claiming to be noether, y500 on cantor gets a fingerprint.

But now what? If y500 on cantor knows noether's fingerprint already then the program can check the values and permit access if they match and deny it if they don't. But what happens if y500 on cantor doesn't know noether's fingerprint? The program y500 is running (slogin in this case) has to ask y500 whether to proceed or not, and all it can show is the fingerprint of the system it is connecting to. So that's what it does.

```
The authenticity of host 'noether.csi.cam.ac.uk (131.111.10.87)' can't be
established.
```

This is the message we saw. The first line explains that y500 on cantor doesn't have a fingerprint for noether to compare against.

```
RSA key fingerprint is 06:d0:42:b2:ea:74:41:c5:49:80:9d:7b:31:44:00:47.
```

Then it tells us what the fingerprint is received was.

```
Are you sure you want to continue connecting (yes/no)? yes
```

And then it asks us whether we want to trust the fingerprint. In this example we said yes.

```
Warning: Permanently added 'noether.csi.cam.ac.uk,131.111.10.87' (RSA) to the list of known hosts.
```

And finally it tells us that it has recorded this accepted fingerprint so that it will never need to ask us this question again.

That's all fine and dandy but how are we supposed to know what the host's fingerprint actually is?

Most people simply trust the fingerprint they see for the first time and cross their fingers. This is not behaviour we should encourage but that's how life is. However, if you have secured access to a system (so you know you are talking to the right system and not a man in the middle) you can ask a system to tell you its fingerprint. For a desktop system this is easy. When you are sitting in front of the system and have just logged in you know what system you are using.

The command to tell you a machine's fingerprint is `ssh-keygen` with the `-l` option:

```
rjd4@noether$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
1024 06:d0:42:b2:ea:74:41:c5:49:80:9d:7b:31:44:00:47
/etc/ssh/ssh_host_rsa_key.pub
rjd4@noether$
```

Note that you have to tell it what file to look in (with the `-f` option). By default it looks for your *personal* private key. We are after the *system's* private key.

If you don't understand the details of this chapter don't worry. You only need to worry about fingerprints the first time you connect to a machine. After that, it's all taken care of automatically.

Exercise

If you have a workstation running the SSH service back in your department or college try connecting to it. You will probably get the full fingerprint challenge. (You will get it unless the system is one of the ones whose fingerprint we record centrally.)

When it goes wrong

Once in a while, the fingerprint of the remote system won't match the local copy. The SSH programs are very verbose about this when it happens:

```
y500@cantor$ slogin rjd4@noether.csi.cam.ac.uk
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!    @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
05:6d:88:6c:8d:1a:76:e2:32:91:8c:6a:29:34:01:bb.
Please contact your system administrator.
Add correct host key in /home/y500/.ssh/known_hosts to get rid of this message.
Offending key in /home/y500/.ssh/known_hosts:2
RSA host key for noether.csi.cam.ac.uk has changed and you have requested
strict checking.
```


Accessing remote Unix systems

```
Host key verification failed.  
y500@cantor$
```

We tried to connect in exactly the same way as we did before, but this time we get a warning message.

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @  
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

It then goes on to give two possible explanations:

```
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!  
Someone could be eavesdropping on you right now (man-in-the-middle attack)!  
It is also possible that the RSA host key has just been changed.
```

The first explanation is that the bad guys are running a man in the middle attack on you.

The second, and far more likely, explanation is that the system administrator has changed the key on the remote system, noether. The usual cause of this is a reinstallation (rather than an upgrade) which has overwritten the previous key files with a new set.

```
The fingerprint for the RSA key sent by the remote host is  
05:6d:88:6c:8d:1a:76:e2:32:91:8c:6a:29:34:01:bb.  
Please contact your system administrator.
```

This is the fingerprint that the machine claiming to be noether is now offering. If noether is not your own machine, approach its system administrators cautiously² because they may be feeling a little defensive about accidentally overwriting the RSA key and ask them if the system has a new RSA key and whether this is the new fingerprint.

```
Add correct host key in /home/y500/.ssh/known_hosts to get rid of this message.  
Offending key in /home/y500/.ssh/known_hosts:2
```

This reveals the innards of the SSH system. The file where the SSH programs keep all their known fingerprints (actually something slightly different) is called `~/.ssh/known_hosts`.³ The file carries one fingerprint per line and the line carrying the fingerprint that you thought was good is line 2 in this example. If it turns out that the system administrator has changed the remote system's RSA key then it is easiest to simply delete line 2 from the file and to then proceed as if it was a "first time" connection.

```
RSA host key for noether.csi.cam.ac.uk has changed and you have requested  
strict checking.  
Host key verification failed.
```

This is the summary of everything that has gone before. The RSA keys don't match and the default behaviour is not to allow you to proceed under these circumstances.

If you get one of these messages contacting a UCS system, please contact the Help Desk urgently (help-desk@ucs.cam.ac.uk, 34681 on the University exchange).

² This is usually demonstrated with gifts of chocolate.

³ Recall that `~` means "your home directory" on Linux and Unix systems.

Transferring files

There are three tools for transferring files between one system and another. The first, `scp`, is simplest and best suited for transferring just a few files or directory trees if you know in advance where they are going to and coming from. The second, `sftp`, is fully interactive and allows for browsing at both the local and remote ends. The third, `rsync`, is a little different. It is designed to keep two sets of files in sync by only transferring the files that need to be transferred.

Exercise

We are going to need some files to illustrate the behaviour of the programs. We will create a directory full of files on our workstations in the `/tmp` directory. Please run the following command on your workstation, replacing the user ID `y500` with whatever ID you are currently using:

```
y500@cantor$ cp -r /ux/Lessons/Remote/data /tmp/y500-data
y500@cantor$ cd /tmp
y500@cantor$
```

Using scp

We will demonstrate the use of `scp` by transferring the directory tree we have just created to a remote system.

```
y500@cantor$ cd /tmp
y500@cantor$ scp -r y500-data rjd4@noether.csi.cam.ac.uk:/tmp
Password: not shown
rfc800.txt          100%   17KB   0.0KB/s   00:00
rfc801.txt          100%   40KB   0.0KB/s   00:00
rfc802.txt          100%   58KB   0.0KB/s   00:00
...
rfc897.txt          100%   15KB   0.0KB/s   00:00
rfc898.txt          100%   41KB   0.0KB/s   00:00
rfc899.txt          100%   39KB   0.0KB/s   00:00
y500@cantor$
```

The location the files are being transferred to is defined by an extension of the “`user@machine`” form used for `slogin`. For `scp` we use the form “`user@machine:location`”. Just as with ordinary `cp` if the location is a directory that already exists then the data is transferred *into* it.

Observe how the files are listed as they are transferred. If the files are large then you get to see a “progress bar” running across the screen as the file is being transferred. All our files are sufficiently small that this really isn't relevant (and nor are the rates of transfer or the ETA of completion).

If we wanted to fetch files from the remote machine in to the local machine we would swap the two final arguments so the remote location would come first and the simple local location would come second.

The `-r` option means “recursive” and has exactly the same effect as it does on plain `cp`. This lets entire directory trees get transferred. If I was sending a file or a list of files then I wouldn't need it. If I had three files called `file1`, `file2` and `file3` then

```
y500@cantor$ scp file1 file2 file3 rjd4@noether.csi.cam.ac.uk:~
```

would transfer them all to `rjd4`'s home directory on `noether`.

Exercise

1. Use `scp` to transfer your `/tmp/y500-data` directory to `/tmp` on the machine `smaug.linux.pwf.cam.ac.uk` (one of the `linux.pwf.cam.ac.uk` servers).
2. Open a second terminal window.
3. In that second terminal window:
 1. `slogin` to `smaug`,
 2. `cd` to `/tmp` and
 3. check that your directory has arrived.

Using sftp

The `sftp` command acts like a primitive `ftp` program. The differences are that `sftp` lacks most of `ftp`'s more sophisticated (and less often used) options and possesses a secure communications link that `ftp` lacks.

To connect to a remote system, issue the command followed by the remote account's name in the same "*user@machine*" format that `slogin` uses.

```
y500@cantor$ sftp rjd4@noether.csi.cam.ac.uk
Connecting to noether.csi.cam.ac.uk...
Password:
sftp>
```

All the options available within `sftp` are listed in response to the `help` or `?` internal command.

```
sftp> help
Available commands:
cd path           Change remote directory to 'path'
lcd path          Change local directory to 'path'
chgrp grp path    Change group of file 'path' to 'grp'
chmod mode path   Change permissions of file 'path' to 'mode'
chown own path    Change owner of file 'path' to 'own'
help             Display this help text
get remote-path [local-path] Download file
lls [ls-options [path]] Display local directory listing
ln oldpath newpath Symlink remote file
lmkdir path       Create local directory
lpwd              Print local working directory
ls [path]         Display remote directory listing
lumask umask      Set local umask to 'umask'
mkdir path        Create remote directory
progress          Toggle display of progress meter
put local-path [remote-path] Upload file
pwd              Display remote working directory
exit             Quit sftp
quit             Quit sftp
rename oldpath newpath Rename remote file
rmdir path        Remove remote directory
rm path           Delete remote file
symlink oldpath newpath Symlink remote file
version           Show SFTP version
!command         Execute 'command' in local shell
!               Escape to local shell
?               Synonym for help
sftp>
```

Accessing remote Unix systems

As can be seen, a small set of the usual commands are available, but these operate on the remote system (noether) rather than the local one (cantor): `cd`, `chgrp`, `chmod`, `chown`, `ls`, `mkdir`, `pwd`, `rm`, and `rmdir`.

Some commands are subtly different. The `exit` command terminates the `sftp` session, which is roughly equivalent to logging you out of the remote system. The `ln` command always creates a symbolic link rather than a hard link. To avoid confusion it is probably better to stick to using the `symlink` synonym.

Some of these remote commands have local equivalents named by preceding them with an “`l`” (for “local”): `lcd`, `lls`, `lmkdir`, `lpwd`, and `lumask`. However, with the exception of `lcd` and `lumask`, these are purely convenience functions. An arbitrary command, other than `cd` or `umask`, can be run by preceding it with a “`!`”. This starts a subshell which runs the command and then exists, returning control to `sftp`.

So the command

```
sftp> lls
```

is exactly equivalent to

```
sftp> !ls
```

and it is possible to issue the command

```
sftp> !date
Wed Apr  5 12:40:20 BST 2006
sftp>
```

where there is no “`ldate`” command within `sftp`.

It is also possible to build up complex command lines after the “`!`” just as you might with the real shell:

```
sftp> !ls | sort -r
```

The two commands where you must use the “`l`” version and not the “`!`” version are `lcd` and `lumask`. The command “`!cd fubar`” would create a subshell and that subshell (and not the `sftp` process itself) would change directory to `fubar`. The subshell would then exist, leaving the parent `sftp` process exactly where it was before. The `umask` command is a more rarely used command. If you know what it does you will understand why the same issue applies.

```
sftp> lcd /tmp
sftp> !mkdir fubar
sftp> lcd fubar
sftp> !ls -l
total 0
sftp> cd /tmp/y500-data/0
sftp> ls -l
-rw-r--r--  1 y500      y500      17764 Apr 10 09:51 rfc800.txt
-rw-r--r--  1 y500      y500      40824 Apr 10 09:51 rfc801.txt
-rw-r--r--  1 y500      y500      59798 Apr 10 09:51 rfc802.txt
-rw-r--r--  1 y500      y500      33110 Apr 10 09:51 rfc803.txt
-rw-r--r--  1 y500      y500      16551 Apr 10 09:51 rfc804.txt
-rw-r--r--  1 y500      y500      12174 Apr 10 09:51 rfc805.txt
-rw-r--r--  1 y500      y500     210138 Apr 10 09:51 rfc806.txt
-rw-r--r--  1 y500      y500      11285 Apr 10 09:51 rfc807.txt
-rw-r--r--  1 y500      y500      15466 Apr 10 09:51 rfc808.txt
-rw-r--r--  1 y500      y500     165483 Apr 10 09:51 rfc809.txt
```

The most commonly used commands in `sftp` are `get` and `put`, the commands used for transferring files.

```
sftp> get rfc806.txt
Fetching /tmp/y500-data/0/rfc806.txt to rfc806.txt
```

Accessing remote Unix systems

```
/tmp/y500-data/0/rfc806.txt          100%  205KB  205.2KB/s   00:00
sftp> !ls -l
total 208
-rw-r--r--  1 rjd4 rjd4   210138 2006-04-10 09:57 rfc806.txt
sftp>
```

These commands can also be used to change the names of files as they are transferred.

```
sftp> get rfc800.txt eighthundred.txt
Fetching /tmp/y500-data/0/rfc800.txt to eighthundred.txt
/tmp/y500-data/0/rfc800.txt          100%   17KB   17.4KB/s   00:00

sftp> !ls -l
total 228
-rw-r--r--  1 rjd4 rjd4    17764 2006-04-10 10:05 eighthundred.txt
-rw-r--r--  1 rjd4 rjd4   210138 2006-04-10 09:57 rfc806.txt
sftp>
```

It can also transfer multiple files, but not rename them.

```
sftp> get rfc80[89].txt
Fetching /tmp/y500-data/0/rfc808.txt to rfc808.txt
/tmp/y500-data/0/rfc808.txt          100%   15KB   15.1KB/s   00:00
Fetching /tmp/y500-data/0/rfc809.txt to rfc809.txt
/tmp/y500-data/0/rfc809.txt          100%  162KB  161.6KB/s   00:00

sftp> !ls -l
total 408
-rw-r--r--  1 rjd4 rjd4    17764 2006-04-10 10:05 eighthundred.txt
-rw-r--r--  1 rjd4 rjd4   210138 2006-04-10 09:57 rfc806.txt
-rw-r--r--  1 rjd4 rjd4    15466 2006-04-10 10:10 rfc808.txt
-rw-r--r--  1 rjd4 rjd4   165483 2006-04-10 10:10 rfc809.txt
sftp>
```

Note that even though the wildcard expanded to two files the sftp program did not mistake this for a renaming get of a single file.

While sftp can transfer multiple files, it cannot transfer directories.

```
sftp> cd ..
sftp> pwd
Remote working directory: /tmp/y500-data
sftp> ls
0 1 2 3 4 5 6 7 8 9
sftp> get 1
Fetching /tmp/y500-data/1 to 1
Cannot download non-regular file: /tmp/y500-data/1
sftp>
```

Even if an expression spans a directory, only the files will be transferred, losing their directory hierarchy in the process:

```
sftp> get 1/*.txt
Fetching /tmp/y500-data/1/rfc810.txt to rfc810.txt
/tmp/y500-data/1/rfc810.txt          100%   14KB   13.9KB/s   00:00
Fetching /tmp/y500-data/1/rfc811.txt to rfc811.txt
/tmp/y500-data/1/rfc811.txt          100%  7771     7.6KB/s   00:00
...
Fetching /tmp/y500-data/1/rfc818.txt to rfc818.txt
/tmp/y500-data/1/rfc818.txt          100%  3693     3.6KB/s   00:00
Fetching /tmp/y500-data/1/rfc819.txt to rfc819.txt
/tmp/y500-data/1/rfc819.txt          100%   34KB   34.5KB/s   00:00
```

```
sftp> !ls -l
total 632
-rw-r--r-- 1 rjd4 users 17764 2006-04-10 10:05 eighthundred.txt
-rw-r--r-- 1 rjd4 users 210138 2006-04-10 09:57 rfc806.txt
-rw-r--r-- 1 rjd4 users 15466 2006-04-10 10:10 rfc808.txt
-rw-r--r-- 1 rjd4 users 165483 2006-04-10 10:10 rfc809.txt
-rw-r--r-- 1 rjd4 users 14196 2006-04-10 10:21 rfc810.txt
-rw-r--r-- 1 rjd4 users 7771 2006-04-10 10:21 rfc811.txt
-rw-r--r-- 1 rjd4 users 5389 2006-04-10 10:21 rfc812.txt
-rw-r--r-- 1 rjd4 users 38110 2006-04-10 10:21 rfc813.txt
-rw-r--r-- 1 rjd4 users 24663 2006-04-10 10:21 rfc814.txt
-rw-r--r-- 1 rjd4 users 14575 2006-04-10 10:21 rfc815.txt
-rw-r--r-- 1 rjd4 users 20106 2006-04-10 10:21 rfc816.txt
-rw-r--r-- 1 rjd4 users 45931 2006-04-10 10:21 rfc817.txt
-rw-r--r-- 1 rjd4 users 3693 2006-04-10 10:21 rfc818.txt
-rw-r--r-- 1 rjd4 users 35314 2006-04-10 10:21 rfc819.txt
sftp>
```

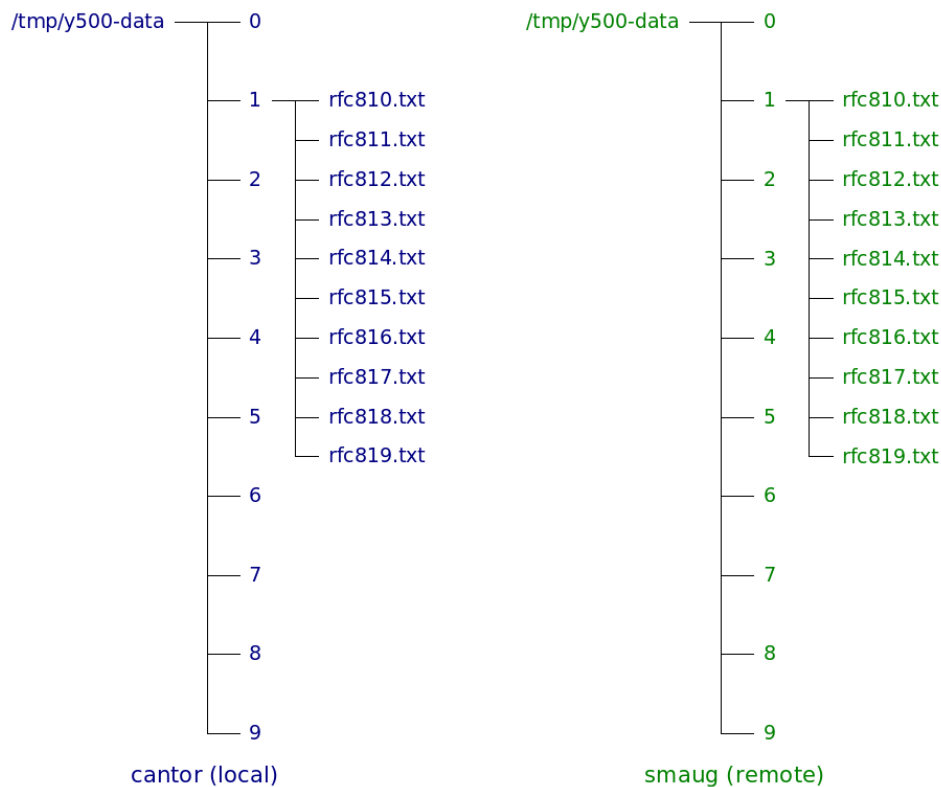
The rsync command

The rsync command was not part of the original *r** commands but followed afterwards. Its purpose in life is to keep in sync sets of files on local and remote systems. Typically a user has a directory tree of files on the two systems and wants to update one set of files (the remote set, say) to reflect the changes made to the other (the local set). The user could simply copy the entire tree from the local system to the remote system overwriting all the files that were present. This would transfer *all* the files, including those that have not been changed. Alternatively the user could manually identify the updated local files and send each of those to the correct location on the remote system.

The rsync program is designed to transmit just the modified files and to do it automatically. (Actually it goes one better. For large files with small changes it identifies just the elements that have changed and sends just those.)

So, on our local system, cantor, we have a directory tree /tmp/y500-data with a copy on smaug. We will use these to illustrate the system.

Accessing remote Unix systems



We will modify just a couple of the local files, `rfc810.txt` and `rfc811.txt`. How we modify them doesn't matter, but for this example we will simply append some text to the end of each file.

```
y500@cantor$ cd /tmp/y500-data/1
y500@cantor$ echo 'Hello, world!' >> rfc810.txt
y500@cantor$ echo 'Hello, world!' >> rfc811.txt
y500@cantor$ cd /tmp
y500@cantor$
```

We now have two files different on the two systems. On the local system, `cantor`, we have the following timestamps and file sizes:

```
y500@cantor$ ls -l y500-data/1
total 224
-rw-r--r-- 1 y500 y500 14210 2006-04-19 09:45 rfc810.txt
-rw-r--r-- 1 y500 y500  7785 2006-04-19 09:45 rfc811.txt
-rw-r--r-- 1 y500 y500  5389 2006-04-19 09:43 rfc812.txt
-rw-r--r-- 1 y500 y500 38110 2006-04-19 09:43 rfc813.txt
-rw-r--r-- 1 y500 y500 24663 2006-04-19 09:43 rfc814.txt
-rw-r--r-- 1 y500 y500 14575 2006-04-19 09:43 rfc815.txt
-rw-r--r-- 1 y500 y500 20106 2006-04-19 09:43 rfc816.txt
-rw-r--r-- 1 y500 y500 45931 2006-04-19 09:43 rfc817.txt
-rw-r--r-- 1 y500 y500  3693 2006-04-19 09:43 rfc818.txt
-rw-r--r-- 1 y500 y500 35314 2006-04-19 09:43 rfc819.txt
y500@cantor$
```

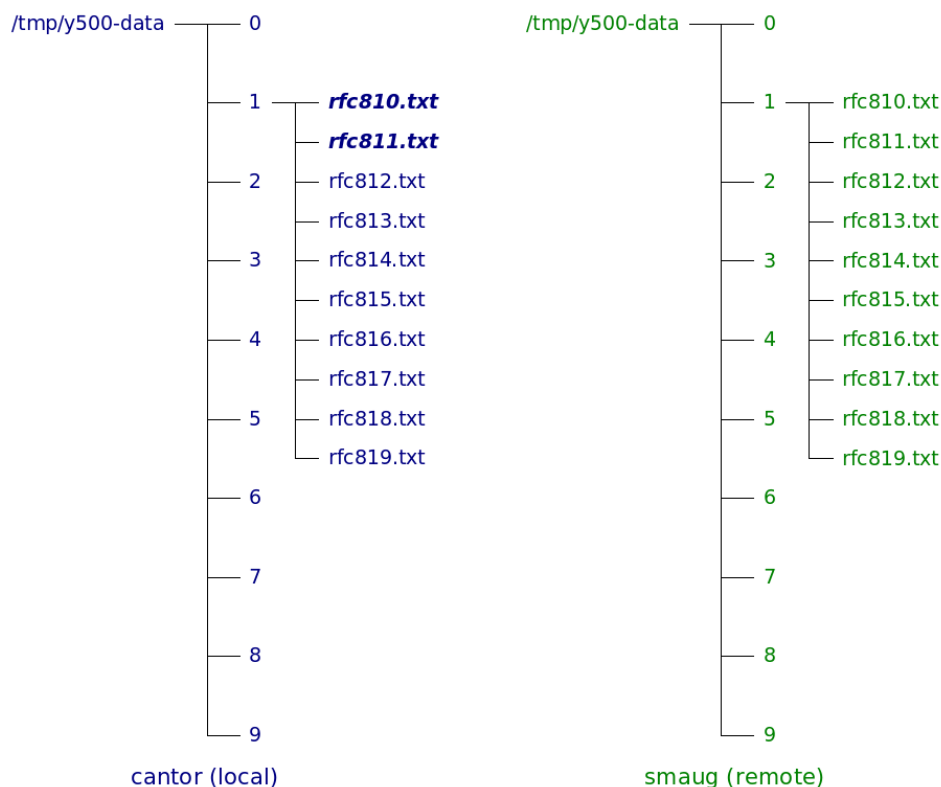
and on the remote system, `smaug`, we have

```
y500@smaug$ ls -l y500-data/1
total 224
```

Accessing remote Unix systems

```
-rw-r--r-- 1 rjd4 rjd4 14196 2006-04-19 09:44 rfc810.txt
-rw-r--r-- 1 rjd4 rjd4 7771 2006-04-19 09:44 rfc811.txt
-rw-r--r-- 1 rjd4 rjd4 5389 2006-04-19 09:44 rfc812.txt
-rw-r--r-- 1 rjd4 rjd4 38110 2006-04-19 09:44 rfc813.txt
-rw-r--r-- 1 rjd4 rjd4 24663 2006-04-19 09:44 rfc814.txt
-rw-r--r-- 1 rjd4 rjd4 14575 2006-04-19 09:44 rfc815.txt
-rw-r--r-- 1 rjd4 rjd4 20106 2006-04-19 09:44 rfc816.txt
-rw-r--r-- 1 rjd4 rjd4 45931 2006-04-19 09:44 rfc817.txt
-rw-r--r-- 1 rjd4 rjd4 3693 2006-04-19 09:44 rfc818.txt
-rw-r--r-- 1 rjd4 rjd4 35314 2006-04-19 09:44 rfc819.txt
y500@smaug$
```

Note how the timestamps on the remote system, *smaug*, are all 09:44, slightly ahead of the timestamps on the majority of files on the local system, *cantor*.



So, how do we synchronise the (now out of date) files on *smaug* with the updated versions on *cantor*?

```
y500@cantor$ cd /tmp
y500@cantor$ rsync --recursive --times --rsh=ssh y500-data
y500@smaug.linux.pwf.cam.ac.uk:/tmp
Password: not shown
y500@cantor$
```

(We will examine that command line blow by blow shortly.)

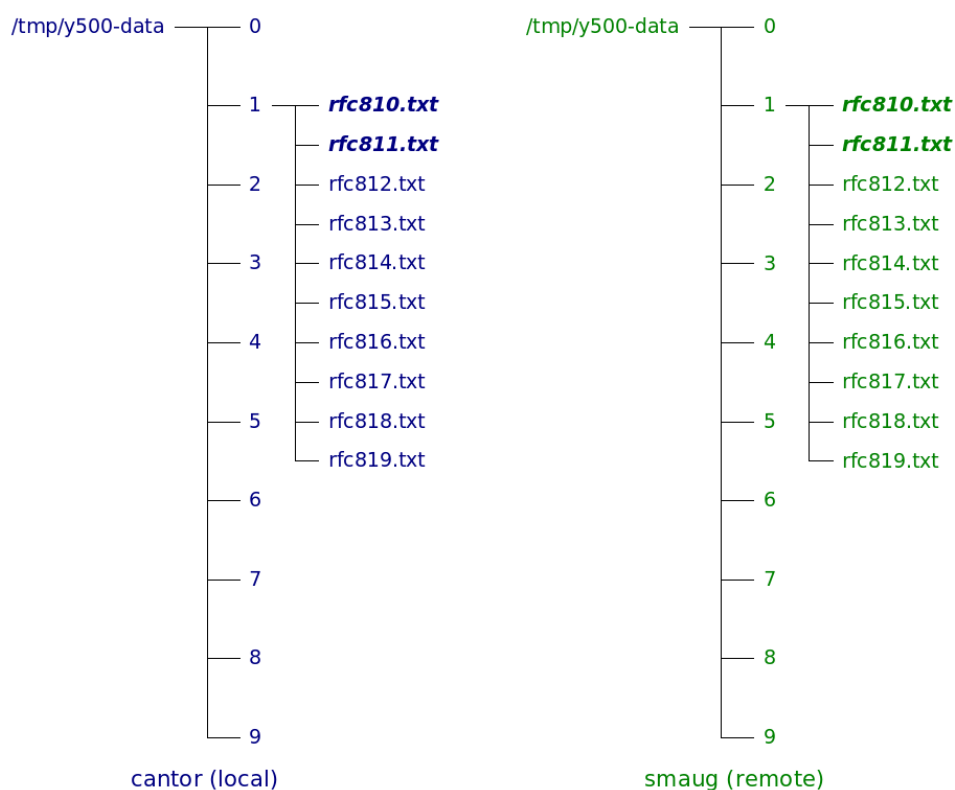
The local set of files is completely untouched by this operation. The remote set of files has been updated in an interesting fashion:

```
y500@smaug$ ls -l y500-data/1
total 224
```


Accessing remote Unix systems

```
-rw-r--r-- 1 rjd4 rjd4 14210 2006-04-19 09:45 rfc810.txt
-rw-r--r-- 1 rjd4 rjd4  7785 2006-04-19 09:45 rfc811.txt
-rw-r--r-- 1 rjd4 rjd4  5389 2006-04-19 09:43 rfc812.txt
-rw-r--r-- 1 rjd4 rjd4 38110 2006-04-19 09:43 rfc813.txt
-rw-r--r-- 1 rjd4 rjd4 24663 2006-04-19 09:43 rfc814.txt
-rw-r--r-- 1 rjd4 rjd4 14575 2006-04-19 09:43 rfc815.txt
-rw-r--r-- 1 rjd4 rjd4 20106 2006-04-19 09:43 rfc816.txt
-rw-r--r-- 1 rjd4 rjd4 45931 2006-04-19 09:43 rfc817.txt
-rw-r--r-- 1 rjd4 rjd4  3693 2006-04-19 09:43 rfc818.txt
-rw-r--r-- 1 rjd4 rjd4 35314 2006-04-19 09:43 rfc819.txt
y500@smaug$
```

Note how the sizes and timestamps of the changed files, `rfc810.txt` and `rfc811.txt`, have been updated to match the local copies. But also note that the timestamps on all the other files have gone *back* to match the local timestamps too. The `rsync` program synchronises timestamps as well as content.



While it isn't obvious from the command line, only data relevant to the changed files was transferred. The `rsync` system compares condensed information about the files at each end to reduce to a minimum the amount of data transferred. Using the `rsync` command, while more complex than `scp` is by far the most efficient way to synchronise sets of files, especially if any of them are large.

We will now consider the various options used on the `rsync` command line to see how we achieved what we have done.

1. `rsync`: This is simply the command we want to run.
2. `--recursive`: This instructs `rsync` to work on directory trees rather than just individual files.

Accessing remote Unix systems

3. `--times`: This is the option that caused the timestamps to be synchronised too. Without it, the timestamps on the remote system would have all been changed to the time of the `rsync` operation.
4. `--rsh=ssh`: This is a slightly weird option. The `rsync` program was originally designed to sit over the insecure `rsh` program. However, it was written at a time when the disquiet about the insecure `r*` programs was rising. Rather than create a separate “`ssync`” program to provide a secure analogue, because it wasn't clear at the time that SSH was going to be the right answer, they defined an option, `--rsh`, with which to specify what alternative to `rsh` the user wanted to use. We choose to use the `ssh` program.
5. `y500-data`: This defines the source files. In our example these are the local files. These files are not changed and the “target” files are updated to bring them into sync with these source files.
6. `y500@smaug.linux.pwf.cam.ac.uk:/tmp`: This defines the “target” files. These are the files that get updated. Swap the local and remote entries to cause a local directory tree to get updated to match a remote one.

There is one other option that might prove useful to you:

7. `--update`: This option stops any updates to the contents of files that would overwrite any target file that claims to be newer than the source file. This can be useful if the files are being updated from more than one source.

Exercise

1. In your editor of choice, modify one of the files in `/tmp/y500-data` on your workstation.
2. Run “`ls -l`” in its directory to check that its time stamp is different from the other files.
3. Wait a minute. (I mean this literally; we want a collection of files with different timestamps.)
4. Edit another file.
5. Run “`ls -l`” again to check its time stamp.
6. Run `rsync` as given above to synchronise the remote set of files on `smaug`.
7. Log in to `smaug` (use `slogin`) and change directory to `/tmp/y500`.
8. Run “`ls -l`” to examine the time stamps.
9. Use your favourite editor to check the content of the two files you modified locally to check that the modifications have transferred.

Doing without the password

So far we have had to quote our password every time we have made a connection. It is possible, under certain circumstances, to set up SSH so that your account on the remote system will trust your account on the local system and not demand a password.

NB: *The PWF Linux servers do not support passwordless access.* You will not be able to follow the examples if you are using `smaug.linux.pwf.cam.ac.uk` as your remote system but only if you are using a more conventional Unix system.

In this worked example, where I will be using a remote system called `noether` and a remote account `rjd4`, we will assume that the SSH implementations are reasonably modern and support the version 2 SSH protocol. SSHv2 supports two different cryptographic systems called RSA and DSA. DSA used to be provided because RSA was encumbered by patent restrictions. This patent has mercifully expired so we can proceed with using pure RSA.

Creating the RSA keys

This stage is done on the local system and need be done only once.

This process creates the thing that will be used to prove that you are who you say you are regardless of passwords. We are going to create private and public data similar to those used to prove a machine is who it says it is, but this time used to prove that a personal account is who it says it is.

```
y500@cantor$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/y500/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/y500/.ssh/id_rsa.
Your public key has been saved in /home/y500/.ssh/id_rsa.pub.
The key fingerprint is:
6f:ee:74:c1:fd:82:c9:ba:13:2c:a9:e2:62:62:56:60 y500@cantor
y500@cantor$
```

What this has done is to create a pair of files. The “private key” file is `/home/y500/.ssh/id_rsa` and must be kept secure. Whoever has access to this file can pretend to be y500. The “public key” file is `/home/y500/.ssh/id_rsa.pub` and should be shared with anyone who wants to see it. Whoever has access to this file can check that anyone who claims to be y500 has access to the private key.

```
rjd4@cantor$ ls -l .ssh/id_rsa*
-rw----- 1 rjd4 rjd4 1675 Apr 19 14:00 .ssh/id_rsa
-rw-r--r-- 1 rjd4 rjd4 407 Apr 19 14:00 .ssh/id_rsa.pub
rjd4@cantor$
```

Transferring the public key

Next I need to copy the public key over to the remote system, `noether`, that is going to trust my local system, `cantor`. This is so that when a connection is made from `cantor` to `noether` claiming to be from y500 `noether` can check that it really is from y500 (or at least from someone with y500's private key).

```
rjd4@cantor$ ssh-copy-id -i ~/.ssh/id_rsa.pub rjd4@noether.csi.cam.ac.uk
26
Password:
```

Now try logging into the machine, with "ssh 'rjd4@noether.csi.cam.ac.uk'", and check in:

```
.ssh/authorized_keys
```

to make sure we haven't added extra keys that you weren't expecting.

```
rjd4@cantor$
```

The ssh-copy-id command takes a public key file, ~/.ssh/id_rsa.pub in this case, and transfers it over to the remote system, noether. On the remote system it adds it to a file called ~/.ssh/authorized_keys. This file, as the name suggests, is the file that lists all the public keys for the accounts that are allowed to connect without quoting a password.

If we connect to the remote system we can see what it has done. Note that we don't get prompted for a password when we make the connection:

```
rjd4@cantor$ slogin rjd4@noether.csi.cam.ac.uk
```

```
Last login: Tue Apr 18 09:21:16 2006
```

```
Have a lot of fun...
```

```
rjd4@noether$ more ~/.ssh/authorized_keys
```

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAQEAme/cePDwuXgJ24zzKCLt1+cK14FyF5ndt24RMvQ
x3iz5Q9NdDJcXr0Ea6Um4i5Fhpy9BTfZAsKrujwdFJckqQJdKP3a1N1kCqMx6VgVUZZosBZ8D
3tQ6oeRfu3NhRq0PGodh/TT0oD8eyya04vqd4IQsFd1qrLw1iJWwk78XLQpMXrCM2uTkTfhBgM
0xQEm90yJYisNfNuGFtSdn0tISkuJ4l5EIgff7E/tHbNLFpkCaQ1zagK10lp7Xvh1aUp4h9oA6
cJxPQTJjTm5T+/hAI+eDH0geczqdulc+cMA7S0RuBg99jQ== rjd4@cantor.csi.cam.ac.uk
```

```
rjd4@noether$
```

Henceforth any connection from y500 on cantor to rjd4 on noether by any of the programs in the SSH suite will be passed without password challenge.

It is important to understand that the "rjd4@cantor.csi.cam.ac.uk" at the end of the line is just a comment. If a miscreant got a copy of my private key file, ~/.ssh/id_rsa from cantor then he or she could connect from any machine in the world to noether and break in to the rjd4 account. The private key file *must* be kept secure.

Exercise

I'm afraid you can only do this exercise if you have access to a "real" Unix or Linux system elsewhere. The PWF Linux servers won't do. Sorry.

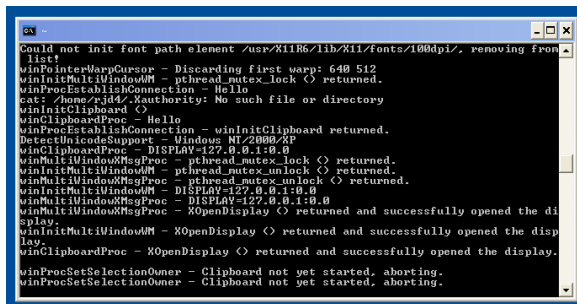
1. Create a local RSA key pair on your PWF account. (ssh-keygen)
2. Transfer the public key over to your remote account so that it permits passwordless access. (ssh-copy-id)
3. Test this by connecting to your remote account and observing that you are not asked for a password. (slogin)

Access to a Unix system from a Windows box

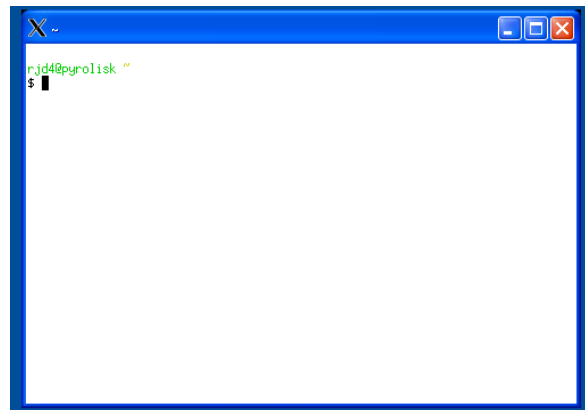
The world does not run on Unix alone but has Windows™ systems too. To access a Unix box from a Windows box it's easiest to start with as Unix as possible an environment on the Windows box. This is best provided by the “CygWin” package which is a collection of the Free Software Foundation's software ported to Windows.

Instructions for installing CygWin on your own PC are outside the scope of this course, but I have installed them under the /ux directory (mapped to X: in PWF Windows) for a demonstration here.

1. Reboot into Windows and log in.
2. Launch a DOS box from the “Start” menu.
3. Issue the command
X:\Lessons\Remote\cygwin\bin\startx
4. A mass of information will scroll past on the DOS box (shown on the left below) and then an X terminal should appear along side it (on the right below). This window can be used in the same way as an ordinary Unix terminal to run `slogin` etc. to connect to a remote Unix system and will display graphical applications also.



```
Could not init font path element /usr/X11R6/lib/X11/fonts/100dpi/, removing from
list
winPointerMapCursor - Discarding first warp: 640 512
winInitMultiWindowWM - pthread_mutex_lock () returned.
winProcEstablishConnection - Hello
cat: /home/rjd4/.Xauthority: No such file or directory
winInitClipboard ()
winClipboardProc - Hello
winProcEstablishConnection - winInitClipboard returned.
DetectUnicodeSupport - Windows NT/2000/XP
winClipboardProc - DISPLAY=127.0.0.1:0.0
winMultiWindowMsgProc - pthread_mutex_lock () returned.
winInitMultiWindowWM - pthread_mutex_unlock () returned.
winMultiWindowMsgProc - pthread_mutex_unlock () returned.
winInitMultiWindowWM - DISPLAY=127.0.0.1:0.0
winMultiWindowMsgProc - DISPLAY=127.0.0.1:0.0
winMultiWindowMsgProc - XOpenDisplay () returned and successfully opened the di
splay.
winInitMultiWindowWM - XOpenDisplay () returned and successfully opened the disp
lay.
winClipboardProc - XOpenDisplay () returned and successfully opened the display.
winProcSetSelectionOwner - Clipboard not yet started, aborting.
winProcSetSelectionOwner - Clipboard not yet started, aborting.
```



If you install CygWin for yourself you get a short cut on your desktop which will launch a DOS box running the CygWin environment for you.

Appendix

This appendix contains some “useful stuff” associated with the course.

SSH fingerprints

This table lists the RSA fingerprints of a number of computers in Cambridge that you might try SSH connections to.

| <i>System</i> | <i>Fingerprint</i> |
|----------------------|--|
| cus.cam.ac.uk | 1024 57:cf:a3:28:4b:ba:58:15:5e:d3:f2:d9:83:72:7c:3e |
| hermes.cam.ac.uk | 1024 2b:7d:58:c6:a2:6b:7f:0d:cf:a8:f9:f7:f8:1d:f9:bb |
| linux.pwf.cam.ac.uk | 1024 74:32:9b:4c:52:47:fd:ad:1b:0e:b8:a7:0f:31:3d:99 |

Write your home/lab/office workstation's host name fingerprint here:

Remember that the command to get the RSA finger print from a local system is:

```
$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
```

On some older systems it might be this:

```
$ ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key
```

as the exact file location can change with versions. Check with your local system administrator⁴ if you can't find it.

Setting your prompt

If you are going to be bouncing backwards and forwards between machines it is worth having your machine name in the prompt the system uses for you to type commands. If you are running bash as your shell (as is the case on most modern Unix systems and virtually all Linux systems) then add the following line to a file called “.bashrc” in your home directory:

```
export PS1='\u@\h\$ '
```

The next time you log in your prompt will be the “user at machine” prompt seen in these notes. If you want to change the current session run that command at the command line.

4 Don't forget the chocolate.