Intermediate SSH Workshop Notes

These notes are based on OpenSSH https://www.openssh.org, installed by default (or readily available) on most platforms (macOS, Linux and Windows). The commands will (mostly) work on the command line for all platforms (Bash, zsh and PowerShell).

Topics:

1. Principles of public key (asymmetric key) cryptography
2. Key Generation and Management
3. Connecting by SSH
4. Using SSH Keys with GitLab and GitHub

1. Principles of public key (asymmetric key) cryptography

This will be covered at the start of the workshop, the most accessible online source is (somewhat unfortunately) Wikipedia:

1. Public-key cryptography https://en.wikipedia.org/wiki/Public-key_cryptography
2. Man-in-the-middle attack https://en.wikipedia.org/wiki/Man-in-the-middle_attack

A very good book for further reading is Simon Singh's *The Code Book* https://simonsingh.net/books/the-code-book which is also available from the University Library https://tewaharoa.victoria.ac.nz/permalink/64VUW_INST/k63216/alma995075654002386.

2. Key Generation and Management

OpenSSH keeps its user files at `~/.ssh` (the `~` means "the user's home directory"). Find the contents of this directory with, for example,

```
ls ~/.ssh
```

There should already be some files, for example:

```
$ ls -1 ~/.ssh
authorized_keys
config
id_ed25519
id_ed25519.pub
known_hosts
```

Creating a New SSH Key Pair

If you don't have a SSH public/private key pair (the `id_*` files in the example above) then create using:

- `ssh-keygen` to generate the key pair,
- `-t` option to select the type of key to create,
- `-C` option to provide a custom comment.

Example:

```
$ ssh-keygen -t ed25519 -C "your.email@vuw.ac.nz"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/username/.ssh/id_ed25519):
Enter passphrase for "/Users/username/.ssh/id_ed25519" (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/username/.ssh/id_ed25519
Your public key has been saved in /Users/username/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:hxyOwoMpkAGFc3lce+hEvyxCKMxyg6m2I8hpBjY7c7E your.email@vuw.ac.nz
The key's randomart image is:
+--[ED25519 256]--+
|+o.o .o          |
|==o.o. +         |
|B=+.. + +        |
|+o * o * +       |
|++o.= + S .      |
|*.= o+ . .       |
|o@ E             |
|+ =              |
|                 |
+----[SHA256]-----+
```

It's wisest to enter a passphrase for your SSH key, and it's best if it's different from your login password.

Warning: if you're prompted

```
/Users/username/.ssh/id_ed25519 already exists.
Overwrite (y/n)?
```

then don't choose `y` to overwrite an existing key pair unless you're sure you no longer need the existing key pair.

Changing the Passphrase or Comment

It's possible to change the passphrase and comment of an existing private key:

- `-p` option changes the passphrase;
- `-c` option changes the comment

of an existing private key file

Example:

```
$ ssh-keygen -p
Enter file in which the key is (/Users/username/.ssh/id_ed25519):
Enter old passphrase:
Key has comment 'your.email@vuw.ac.nz'
Enter new passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved with the new passphrase.
```

Fingerprints

Check fingerprints (via another channel) whenever practicable! Use:

- `ssh-keygen -l` to output the fingerprint,
- `-E` option to specify a hash algorithm other than the default,
- `-f` option to specify a target file.

Example: showing the MD5 fingerprint of a user's key

```
$ ssh-keygen -E md5 -l -f ~/.ssh/id_ed25519.pub
256 MD5:a2:f3:54:b1:6a:b7:9b:87:a5:be:aa:2d:6c:c6:0b:f7 your.email@vuw.ac.nz (ED2551
```

When verifying a fingerprint, you may need to specify the hash algorithm to match the fingerprint presented.

SSH Agent

Up to this point, ssh will request your private key passphrase on every use, which quickly becomes tiresome. Fortunately, theres a service called ssh-agent which will hold private keys for authentication.

Adding

The command to add a private key to the agent is:

```
ssh-add
```

which will prompt for the passphrase of the private key (see *Starting ssh-agent* below if this doesn't work). After entering the passphrase once, it won't need to be entered again for the duration of the session.

The plain `ssh-add` command will add every SSH key in `~/.ssh/` to the agent. To add a specific key, specify the path to the private part of the key, for example:

```
ssh-add ~/.ssh/id_ed25519
```

You can check which identities are stored in the agent by using the `-l` flag

```
$ ssh-add -l
256 SHA256:hxyOwoMpkAGFc3lce+hEvyxCKMxyg6m2I8hpBjY7c7E your.email@vuw.ac.nz (ED25519
```

Use the `-E` flag to specify the hash algorithm used when displaying key fingerprints.

```
ssh-add; ssh-add -E md5 -l
```

Starting ssh-agent

If you receive an error message

```
$ ssh-add
Could not open a connection to your authentication agent.
```

then you'll need to start ssh-agent. On macOS and Linux this can be accomplished with:

```
eval $(ssh-agent)
```

On Windows you'll need to enable ssh-agent and start the service, see the PowerShell instructions in the appendix *Installing OpenSSH on Windows*.

3. Connecting by SSH

Thus far everything has been local, but the purpose of SSH is to connect to remote hosts.

Connecting to a Remote Host

Hosts have SSH keys, just like users. The location of these keys is (unfortunately) platform-dependent, on macOS and Linux the host keys are typically found in `/etc/ssh/`.

```
$ ls /etc/ssh/
moduli                          ssh_host_ed25519_key
ssh_config                      ssh_host_ed25519_key.pub
ssh_config.d                    ssh_host_rsa_key
ssh_host_dsa_key                ssh_host_rsa_key.pub
ssh_host_dsa_key.pub            sshd_config
ssh_host_ecdsa_key              sshd_config.d
ssh_host_ecdsa_key.pub
```

Find a host's fingerprint by running `ssh-keygen -l`, for example:

```
$ ssh-keygen -E md5 -l -f /etc/ssh/ssh_host_ecdsa_key.pub
256 MD5:d6:a1:72:ce:70:38:14:64:55:b8:5e:35:a6:cd:34:cb username@host (ECDSA)

$ ssh-keygen -E md5 -l -f /etc/ssh/ssh_host_ed25519_key.pub
256 MD5:35:ee:49:a1:6c:ec:3d:ec:a4:cd:89:f5:a9:6c:f9:a0 username@host (ED25519)
```

When first connecting to a host, ssh will prompt for fingerprint verification. For example, connecting to Raapoi for the first time:

```
$ ssh username@raapoi.vuw.ac.nz
The authenticity of host 'raapoi.vuw.ac.nz (130.195.19.126)' can't be established.
ED25519 key fingerprint is SHA256:f+rhB7q5nt/HxcNK3qA8UfSdSJ7J05L1dU4C2fslkxg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

It's important to verify the fingerprint to avoid falling victim to a machine-in-the-middle attack. The Raapoi Cluster Documentation https://vuw-research-computing.github.io/raapoi-docs/accessing_the_cluster/#ssh-clients has the fingerprint and advises:

> The first time an SSH client connects to the server, it displays the servers public key fingerprint. An SSH host key identifies the server to your ssh client. They are an important security feature and not something you should just hit ENTER to accept.

If you have physical (or other) access to the host you're connecting to by SSH then you can generate the host key fingerprint on the command line with the commands above. Once satisfied with the authenticity of the host public key presented, enter `yes`, after which a password will be requested:

```
$ ssh username@raapoi.vuw.ac.nz
The authenticity of host 'raapoi.vuw.ac.nz (130.195.19.126)' can't be established.
ED25519 key fingerprint is SHA256:f+rhB7q5nt/HxcNK3qA8UfSdSJ7J05L1dU4C2fslkxg.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'raapoi.vuw.ac.nz' (ED25519) to the list of known hosts.
username@raapoi.vuw.ac.nz's password:
```

Known Hosts

When a host's public key authenticity is verified, i.e. `yes` is entered to the query above, the public key is added to `~/.ssh/known_hosts` as indicated in the command output. This file is a plain text file containing host names (or IP addresses if there's no host name) followed by the public key.

To see the fingerprints of all public keys in `known_hosts`, use the `ssh-keygen` command:

```
256 SHA256:f+rhB7q5nt/HxcNK3qA8UfSdSJ7J05L1dU4C2fslkxg raapoi.vuw.ac.nz (ED25519)
256 SHA256:jg0wcBnkXxm1D0RxgBmmON9KApaMXgbcWGnuDKnom1w gitlab.ecs.vuw.ac.nz (ED2551
256 SHA256:p2QAMXNIC1TJYWeIOttrVc98/R1BUFWu3/LiyKgUfQM github.com,20.248.137.48 (ECD
256 SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU github.com (ED25519)
...
```

ssh-keygen provides a couple of options for maintaining the `known_hosts` file:

- `-F <host>` option will search `known_hosts` for entries for `<host>`
- `-R <host>` option will delete entries for `<host>` from `known_hosts`

Examples:

```
$ ssh-keygen -F raapoi.vuw.ac.nz
# Host raapoi.vuw.ac.nz found: line 1
raapoi.vuw.ac.nz ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPtbSi7YWx0SQ24cF83pyQ0CwIVzLGC

$ ssh-keygen -R raapoi.vuw.ac.nz
# Host raapoi.vuw.ac.nz found: line 1
/Users/username/.ssh/known_hosts updated.
Original contents retained as /Users/username/.ssh/known_hosts.old
```

Setting-up Public Key Authentication

Up to now it's been possible to connect to a remote host with password, but who needs the toil of entering a password? We can eliminate this friction from SSH connections by setting-up public key authentication (which is more secure than password authentication).

Authorised Keys

`~/.ssh/authorized_keys` are the public keys of key pairs which are allowed to authenticate for this user. Adding a public key to `~/.ssh/authorized_keys` allows a remote user to authenticate to the account without password, i.e. by public key authentication.

Now, we have a key distribution problem.

We can manually copy keys (below), but on Bash or zsh it's more convenient to use a utility called ssh-copy-id

> ssh-copy-id is a script that uses ssh to log into a remote machine... By default it adds the keys by appending them to the remote user's `~/.ssh/authorized_keys`

Example:

```
ssh-copy-id username@raapoi.vuw.ac.nz
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: ssh-add -L
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter out
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are prompted r
username@raapoi.vuw.ac.nz's password:
```

Note the automatic use of ssh-agent by this command.

Manually Copying Keys

On Windows a PowerShell command can be run to copy the local public key to the remote host

- Windows 10 OpenSSH Equivalent of ssh-copy-id https://chrisjhart.com/Windows-10-ssh-copy-id

Similarly, it's possible to manually copy a key on Bash or zsh with a command

- Easiest way to copy ssh keys to another machine? https://askubuntu.com/questions/4830/easiest-way-to-copy-ssh-keys-to-another-machine

Connecting from Outside the University

Sometimes we don't have direct access to the remote host, we need to connect to a gateway first.

All staff: Connecting your local machine to the University VPN will allow a direct SSH connections to the remote host.

ECS and SMS staff: we can first SSH to `entry.ecs.vuw.ac.nz` and then SSH to the remote host.

Again, who needs the toil of manually connecting when we can have SSH do the work for us by specifying a "jump host"?

Jump Hosts

Connect to a remote via an arbitrary number of intermediates, using the `-J` flag. Example connecting to a host which has only an IP address:

```
ssh -J entry.ecs.vuw.ac.nz username@10.140.xx.yy
```

SSH Configuration

There are lots of useful options which can be configured, for example jump hosts and host names. Create a `~/.ssh/config` file to define hosts (see: `man ssh_config`). Example:

```
Host MyPC
  ProxyJump entry.ecs.vuw.ac.nz
  Hostname 10.140.xx.yy
  User username
```

This allows connection by `ssh MyPC` and public key authentication functions transparently. No more manual toil!

This also works with `scp`, no more copying files from remote to local with:

```
scp -p -J entry.ecs.vuw.ac.nz 10.140.xx.yy:<source> <destination>
```

instead we can write

```
scp MyPC:<source> <destination>
```

Using SSH Keys with GitLab and GitHub

SSH keys can be used with Git so repositories can be cloned with SSH. Combined with `ssh-agent`, this avoids the problem with cloning by HTTPS: having to re-enter your password on the command line for access to the remote repository.

Documentation

Both GitLab and GitHub have great documentation on how to configure SSH access:

1. Use SSH keys to communicate with GitLab https://docs.gitlab.com/ee/user/ssh.html
2. Adding a new SSH key to your GitHub account https://docs.github.com/en/authentication/connecting-to-github-with-ssh/adding-a-new-ssh-key-to-your-github-account

Host Key Verification

You'll be asked to verify the host key fingerprint on first use, which are published on the GitLab and GitHub web pages:

1. Raapoi Cluster https://vuw-research-computing.github.io/raapoi-docs/accessing_the_cluster
2. ECS GitLab Instance Configuration https://gitlab.ecs.vuw.ac.nz/help/instance_configuration
3. GitLab.com Instance Configuration https://gitlab.com/help/instance_configuration
4. GitHub's SSH key fingerprints https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/githubs-ssh-key-fingerprints

Switching an existing clone from HTTPS to SSH

On the command line, with the working directory in the cloned repository, switch the remote URL to the SSH URL (`git@...`) from the repository web page with:

```
git remote set-url origin git@...
```

You only need to do this if the existing remote URL is HTTPS (`https://...`), which you can check with:

```
git remote -v
```

Appendix: Installing OpenSSH on Windows

OpenSSH is bundled with Windows but not installed by default. It's an easy process to install, you'll require administrator privileges for the installation. For this workshop, install OpenSSH into PowerShell:

1. Get started with OpenSSH for Windows https://learn.microsoft.com/en-us/windows-server/administration/openssh/openssh_install_firstuse?tabs=gui&pivots=windows-11
2. How To Install The OpenSSH Client On Windows https://www.itechtics.com/openssh-for-windows

Activating ssh-agent

Unfortunately, ssh-agent isn't enabled or running by default after installing OpenSSH. You'll need

administrator privileges to start the service so you can use the `ssh-agent` command in PowerShell:

1. Starting ssh-agent in Windows PowerShell https://peateasea.de/starting-ssh-agent-in-windows-powershell
2. Starting ssh-agent on Windows 10 fails: "unable to start ssh-agent service, error :1058" https://stackoverflow.com/questions/52113738/starting-ssh-agent-on-windows-10-fails-unable-to-start-ssh-agent-service-erro

Copyright

Authors: James Quilty
Affiliation: School of Engineering and Computer Science, Victoria University of Wellington
Last updated: 2025-12-04