PART1-WEEK1-REVIEW

Shell command

POSIX Systems

Slides- Secure shell

SSH servers have a key pair:

- connect to new server → asks you if you're sure
- connect to a server & key has changed → get warning

Keys needs to be convenient and secure.

Servers: bastion host → load balancer → (real) lab machines

The bastion host acts as the entry point at the network perimeter, receiving requests from clients that may need access to services hosted on multiple backend machines. **Load balancing** is responsible for distributing these requests to multiple backend machines to achieve load balancing, ensuring a relatively even distribution of workload among them to enhance system performance and availability. They work together to ensure that requests from clients are efficiently processed and distributed to multiple backend machines.

Exercises

1. **Secure shell**
    - Is a protocol to allow you to remotely log in to another computer.
    - (OpenSSH) is actually two programs:
    1. ssh - **client** - run on own machine to connect to another machine.
    2. sshd - **server** or daemon in UNIX-speak - runs in the background on the machine you want to connect to.
    3. default TCP port SSH:22.

```
ssh localhost
# will happen four situations
# 1.ask for a password -> ssh server is running on current machi
# 2.succeeds without a pw -> client is working + server is runni
#   your machine, or already have a key set up
# 3.connection refused -> ssh client correctly working, but no s
#   on you own machine.
# 4. error or not found: don't have (Open)SSH installed.
```

bastion host: This is reachable over SSH from the internet, and is on a university network that lets you connect further to the lab machines.

load balancer: This connects you to a lab machine that is currently running and ensures that if everyone uses this method to connect, then they will be more or less equally distributed among the running machines.

```
# command 'uname -a' to print message
# uname - print the information about system
[vz23211@it075831 ~]$ uname
Linux
# -a : shows all information
[vz23211@it075831 ~]$ uname -a
Linux it075831.wks.bris.ac.uk 4.18.0-513.11.1.el8_9.x86_64 #1 SM
# whoami: shows university username
[vz23211@it075831 ~]$ whoami
vz23211
# hostname: shows the machine name
[vz23211@it075831 ~]$ hostname
it075831.wks.bris.ac.uk
# exit - exit the connection
```

`ssh -J USERNAME@seis.bris.ac.uk USERNAME@rd-mvb-linuxlab.bristol.ac.uk`

The `-J` for "**jump through this host**" even accepts a comma-separated list of hosts if you need to connect through more than one. However, you need to **repeat your username** for every machine.

SSH KEYS

connect to machine - log in method

Three main authentication factors: 1.something you know(password or PIN). 2.something you have(key, ID card, passport). 3.something you are(biometrics)

- log in with a username & password(1)
- log in with a digital key(2)
- log in with a key file that is itself with a password. → two-factor authentication

KEY - private key & public key

一般自己的机器上都是私钥（没有后缀）。server的机器上都是公钥（有.pub）

- private key: a file normally named `id_CIPHER` - keep it store place **own you** have access to
- public key: a file normally named `id_CIPHER.pub` - can share it to **world**
- The `-t` parameter selects the cryptographic algorithm to use.

```
rw-------. 1 vagrant vagrant 411 Oct 7 10:50 id_ed25519 # privat
-rw-r--r--. 1 vagrant vagrant 98 Oct 7 10:50 id_ed25519.pub #pub
-rw-r--r--. 1 vagrant vagrant 1597 Oct 7 11:54 known_hosts
```

File permission simple(简易文件权限)

example: (-)(rw-)(r--)(r--)

r-read w-write x-execute

- first bracket - only applies to special file types(d for directory)
- second bracket - owner permission - 3bits
- third bracket - group permission -3bits
- forth bracket - everyone else -3bits

`known_hosts` is where SSH stores the public keys of computers you've already connected to. The file format is one key per line and you can edit the file yourself if you want to. It stores on own machine (client).

The command for copying a file is `scp` for secure copy, which works like `cp` but allows you to include remote hosts and does the copy over SSH. Run this from your own machine.

`scp source destination` - in here in order to copy the public key to the server.

```
cd .ssh
cat id_ed25519.pub >> authorized_keys
chmod 600 authorized_keys
```

SSH will accept a public key if it is listed in the file `authorized_keys` in the user's `.ssh` folder, the format is one line per key. We use the construction `cat SOURCE >> DEST` to have our shell append the source file to the destination.

cat SOURCE >> DEST : append

`Chmod` command - change permissions/mod bits

`chmod 600 file_name` - owner: 6(decimal) → 110(binary) → first three bits is owner , read & write permission

example `chmod XXX filename` : chmod 755(八进制) 就是chmod 111 101 101(binary) 因为位数是read/write/execute 这三个三位bits的顺序是owner group other 所以权限就是owner rwx， group rx， other， rx

u → owner; g → group; o →other; a →all;

+: add permission; -: remove permission; = : set permission equals to

`chmod u+x` file.txt: Adds execute permission for the owner.

`chmod go-w` file.txt: Removes write permission for the group and others.

`chmod a=r` file.txt: Sets read-only permission for all users.

ssh doesn't work → add `-v` switch enable **debugging** information(`-vv` or `-vvv` see more detail)

man - manual

`man command` can open the command's manual page. **Important to exam!**

For example : `man git` can open the manual page of git which is very useful.

`man ssh_config | less` less can show one page a time, lets you scroll and search touch - create a empty file or update the access and modification timestamps of an existing file. `touch a1.txt` → create one empty text file called a1.

different keys - `-i FILENAME` to select a private key file.

Slides- Vagrant

- start the machine: `vagrant up` : default: 22 (guest) ⇒ 2222 (host)
- log in: `vagrant ssh`
- stop machine: `vagrant halt`
- stop+ start machine: `vagrant reload`
- delete machine: `vagrant destroy`

All commands require a Vagrantfile in the current directory.

Storage: Linux: ~/.vagrant.d ; Windows: C:\Users\NAME\.vagrant.d

EXERCISES

The `<<-SHELL` construction is called a "here document", and is a way in some programming languages of writing multi-line strings

Running Vagrant and shutting down cleanly

`vagrant up` → `vagrant ssh` →In vagrant: coding → In vagrant: exit → `vagrant halt`

Slides- Package managers

ALL APT(package manager) command →

- finding packages

`apt search [-v] [-d] STRING`

`apt info [-a] PACKAGE`

`apt list [-I] PACKAGE`

`apt [COMMAND] --help`

- update and upgrade

`sudo apt update` - download new lists of packages but **don't install anything yet**

`sudo apt upgrade` - upgrade all **installed** packages to the latest version

- Installing

`sudo apt install PACKAGE [PACKAGE...]` - Installs one or more packages

`cat /vagrant/Vagrantfile`

```
vagrant@debian12:~$ cat /vagrant/Vagrantfile
Vagrant.configure("2") do |config|
  config.vm.box = "generic/debian12"
  config.vm.synced_folder ".", "/vagrant"

  config.vm.provision "shell", inline: <<-SHELL
    apt-get update -y
    apt-get install -y git git-man apt-file
    apt-file update
  SHELL
end
```

- Finding a command - which package provides the complete path of currently search command.

`dpkg-query -S /bin/bash` — need a complete file path.

- 提供完整路径，返回哪个安装包提供了你写入的完整路径 e.g.

```
vagrant@debian12:~$ dpkg-query -S /bin/ls
coreutils: /bin/ls
vagrant@debian12:~$ dpkg-query -S /bin/bash
bash: /bin/bash
```

EXERCISES

The file system

`/bin` stands for binaries, that is **programs** that you can run.

`/usr` and its subfolders are for normally **read-only data**, such as **programs and configuration files** but not temporary data or log files.

`/etc` stores **system-wide configuration** files and typically **only root** (the administrator account) can change things in here.

`/home` is the folder **containing users' home directories**, for example the default user vagrant gets `/home/vagrant`.

`/lib` contains dynamic libraries.

`/sbin` (system binaries) is another collection of programs, typically ones that **only system administrators** will use.

`/tmp` is a **temporary filesystem** that may be stored in RAM instead of on disk.

`/var` holds files that vary over time, such as logs or caches.

`/dev`, `/sys` and `/proc` : **virtual file system**s.

`/dev` offers an interface to devices such as hard disks, memory and a number of pseudoterminals or ttys.

`/proc` provides access to running processes;

`/sys` provides access to system functions.

The `/vagrant` folder is **not part of the FHS**, but is our convention for a shared folder with the host on Vagrant virtual machines.

Find out where a program is with `which`, so `which ls` will show you `/usr/bin/ls`

`/bin` was only for binaries needed to start the system -most **important** binaries.

`/usr/bin` was where most binaries lived which were available **globally**.

`/usr/local/bin` was for binaries installed by a **local** administrator.
Default colors: file type: green is an executable program, blue is a link to another file.

```
root@debian12:~# ls -l /bin
lrwxrwxrwx 1 root root 7 Jan  9 20:10 /bin -> usr/bin
```

(First)Main ones being `-` for normal file, `d` for directory and `l` for a so-called *soft link*.
In code above, l means it's one soft link type.

Package managers

`nano FILENAME` - edit file

`sudo apt install PACKAGE`

- `sudo` - allow to run command as root
- `apt` - package manager
- `install` - add one package(download a lot of things)
- `PACKAGE` - package name

`sudo apt remove PACKAGE` - removes don't like package from system

Lab machine → in vagrant file add command like `apt install PACKAGE` below line `echo` to list as many package as user needs. There is no `sudo` here because when Vagrant is **installing** the system, it is running as **root automatically.**

Soft link(Symbolic link) & hard link

Symbolic link

A symbolic link, also known as a symlink or soft link, is a special type of file that points to another file or directory. Using the `ln -s` command to create.

Hard link

A hard link is another name for a file on a file system, essentially creating multiple directory entries (file names) that point to the same file data. Actually they are the same file who has different filename. Using the `ln` command to create.

Summary

实际上，硬链接和源文件是同一份文件，而软连接是独立的文件，类似于快捷方式，存储着源文件的位置信息便于指向。 使用限制上，不能对目录创建硬链接，不能对不同文件系统创建硬链接，不能对不存在的文件创建硬链接；可以对目录创建软连接，可以跨文件系统创建软连接，可以对不存在的文件创建软连接。

Slides- The Shell & Shell expansion

Terms: shell, terminal, console, command line, (command) prompt

- Shell workflow: User request shell, shell response user.
- Prompt: $(in a shell); #(in a root shell); %(probably in the C shell);

  >(on a continuation line)

shell tricks:

`TAB`: complete command or filename

`DOUBLE TAB`: show list of possible completions

`UP/DOWN`: scroll through history (方向键的上下)

`^R text`: search history for command(ctrl + R)

- builtins： `which ls` → result `/bin/ls`

`ls` basic `ls -l` details `ls -a` do not ignore entries starting with .

help & manuals

- `COMMAND  --help` for example: `ls --help` , `git --help` , `apt --help`
- `man COMMAND` for example: `man git` , `man ls`
- `man 1 printf` and `man 3 printf` are different.

Section 1,2,3: Different contains different types in manuals.

Section 1: contains common **shell commands**.(can directly input in terminal)

Section 2: contains **System Calls.**

Section 3: contains **C Library Functions**.

Shell deals with expanding **pattern; programs** deals withs its **arguments**.

***** : **all** filenames in **current scope**

?: single character in filename → e.g. **image???.jpg** matches **image001.jpg**

[ab]: single character in list → e.g. image[0-9].jpg

$: variable name expansion

```
name="John"
echo $name
```

“double quotes”: turn off pattern matching keeps variable interpolation and backslashes on.

‘single quotes’: turn of everything

***, \?, \[, \$** : do not treat as pattern（不要解释成别的意思，就是原本的意思）

CP

`cp [-rfi] SRC... DEST copy files`

-r recursive; ***-f:*** overwrite read only ***-i:*** ask before overwriting (interactive)

`mv [-nf] SRC... DEST move files`

- ***-n*** no overwrite , ***-f*** force overwrite

Find files

`find DIR [EXPRESSION]` **-** `find . -name “a*”`

- `find . -name “a*”`

EXERCISES

Whitespace - 空白

How shell handle whitespace in line?

- no double quote “” → as different arguments
- has double quote “” → as one argument

Files name **has spaces**: Without quotes, the shell will interpret **each word separated** by spaces as a separate argument.

Shell variables - good to double quote “” to prevent silly name with spaces

You need to set the **variable** on a **separate line**.

`VARIABLE=VALUE` sets a variable to a value

`$VARIABLE` retrieves its value.

`${a}b` means the value of the variable `a` followed by the letter b

`$ab` would mean the value of the variable `ab`: needs to set ab first. e.g. `ab=dasjlda`

```
# correct using silly name
program="silly name"
gcc -Wall "$program.c" -o "$program" # double quote -> execute a
# If write as: 1. set value silly 2. execute program name
program=silly name
```

Slides- Pipes(1&2 together)

Unix Philosophy:

1. Each program should do one thing well.
2. Programs should be able to cooperate to perform larger tasks.
3. The universal interface between programs should be a text stream.

输入/输出的way有三种不同的形态：

- 文件名[FILENAME]
- 标准流 (0,1,2)
- 作为参数输入(arguments)

Pipes（|）→使用的是标准流来在命令之间传输输入和输出结果

redirect（<,>，>>(appends)） → 一般是把东西输出到文件里，所以参数往往是FILENAME

COMMAND $(SOMETHING) → 这种就是作为arguments输入一个东西

standard IO - IMPORTANT

- 0 = standard input
- 1 = standard output
- 2 = standard error

```
./example.sh > output.txt 2> error.txt
```

`>&2` 是一种输出重定向语法，用于将输出发送到标准错误（stderr）而不是标准输出（stdout）。

在shell中， `>` 符号通常用于将命令的输出重定向到文件中。例如， `command > file.txt` 将命令 `command` 的输出发送到 `file.txt` 文件中。但是，当你想将输出发送到**标准错误**而不是标准输出时，你可以使用 `>&2` 语法。

在 `echo "Oops! Something went wrong." >&2` 这个例子中， `>&2` 将 `echo` 命令的输出重定向到文件描述符 2，即标准错误。这意味着消息 "Oops! Something went wrong." 将会被发送到标准错误流中，而不是标准输出流中。

`>` and `>&`

- `>` 符号用于将命令的标准输出重定向到指定的位置，通常是文件.

  `command > filename` **e.g.** **`echo "Hello" > output.txt`**

- `>&` 符号用于将一个文件描述符的输出重定向到另一个文件描述符。

  `command >& file_descriptor` **e.g.** **`command >&2`**

Pipes - one command output pipes to next command input

- `ls -l | head [-n](number)` or `ls -l | tail [-n](number)`

  Lists numbers of **head** Lists numbers from **tail**

`|` → called pipes, command & results will be connect from this pipes.

- `$ ls -1 | grep software | sort -r`

  Explain: `grep`: "global regular expression parser”

  `sort:` read all lines into buffer, sort, output.

  `uniq`: remove duplicates immediately following.

  Best used as: `command | sort | uni`

  Reason: `sort` first → duplicates next to each other → `uni` most effective way to utilize(use) the `uniq` command.

- ***grep***

`grep PATTERN FILENAMES`-search for a specified pattern (`PATTERN`) in the specified file(s) (`FILENAMES`).

`grep -nHi PATTERN FILENAMES`

- The `n` option is used to print the **line numbers** of matching lines.
- The `H` option is used to **print** the **filenames** of matching lines (if more than one file is searched).
- The `i` option is used to **ignore case sensitivity**.
- Example: `grep -nHi "hello" file1.txt file2.txt`

`grep [OPTIONS] PATTERN`-waits for the user to input text from **standard input** and then searches for the specified pattern within it.

E.g. `ls -l | grep "file"`

- ***sort - read** the lines in **buffer**, sort, output*

Redirects

`cat infile | sort > outfile` : shows the "infile" context in buffer, sort the buffer then redirecting into the outfile(If not have this file, then create it auto).

` sort < infile > outfile`: It has same result with `cat infile | sort > outfile`

`<`and`>`:
This symbol (
`<`) is used to redirect input while This symbol (`>`) is used to redirect output.

`>`**and** `>>`**:** `COMMAND > FILE` : **overwrite** file ; `COMMAND >> FILE`: **appends to** file
*
error redirect*

`COMMAND > FILE 2> FILE2` : standard output redirects to FILE, then standard error redirects to FILE2.

`COMMAND FILE 2>&1` : standard output and standard error redirect to **same** FILE

*标准重定向一般不能有空格

ignore output : `COMMAND > /dev/null`

Files VS Streams - can convert between each other

Streams and by instead by **files**. (Put all needed information into File.)

- `PROGRAM < FILE` (standard input) - file contains input information
-

`PROGRAM > FILE` (standard output) - file contains programs output result

-

`PROGRAM 2> FILE` (standard error) - files contains programs standard error output

??? programs < FILENAME can use standard I/O instead by.

—标准输出和输入，怎么切换形式打开, 答案是加dash-

slash : / ; dash: -:

以dash(-)开头的坏文件名，要用 command /-.FILENAME 这种命令去打开

Filenames with dashes(-): BAD Solution when addressing one: `cat ./-` or `rm ./-f`

Advanced

- `ls | tee FILE`

  takes a **filename** as argument and writes a **copy** of input to it, as well as to **stdout**.

- `ls | less` : pager: it displays text on your screen, one page at a time.

- `sed` : `echo “Hello world” | sed -e ‘s/World/Universe/’` → Hello Universe

  Changes text using a regular expression as it passes from input to output ← The middle

  Replace: `s/ONE/TWO/[g]` replaces the first match for ONE (all matches, with /g) with TWO. Regular expressions are supported.

- `PROGRAM` `<(` `SOMETHING)` - program wants a file to read from. **pipes** something in.

  `<(` NO SPACE HERE! *不能有空格！

```
vagrant@debian12:~$ cat <(echo "Hi")
Hi
# 路径指向一个匿名 FIFO 文件，其中包含了 echo "Hi" 命令的输出
vagrant@debian12:~$ echo <(echo "Hi")
/dev/fd/63
```

```
# /dev/fd/63 是一个特殊的文件路径，代表着一个文件描述符
# <(command) 的语法会创建一个匿名的 FIFO 文件（也称为命名管道），
# 然后将命令 command 的输出重定向到该 FIFO 文件中，并返回该 FIFO 文
```

subshell

`$ COMMAND $(SOMETHING)` ← e.g. 这就是以arguments输出东西！

```
vagrant@debian12:~$ echo $(echo "Hi" | sed -e 's/Hi/Hello/')
Hello
```

Exercise

- `ls -head` **- default first 10 lines(Same as the tail command)**
- `cat [FILENAME [FILENAME….]]` **：显示到shell (standard output)**
- `head [-n N]` **&** `tail [-n N]` **：头、尾**

  e.g. `head -n -2` to *skip the last 2 lines* and write all the rest.
- `sort` **&** `uniq` **：排序、去重**

  A common way to remove duplicate lines is `... | sort | uniq | ...`.

```
vagrant@debian12:~$ cat review hello.c
Today is 24-2-2024, I don't want to review, but I have to do tha
Cause I don't wanna fail two course( need to delay graduate whic
big challenge of my money and mood.
#include <stdio.h>
int main(){
  printf("Hello from Amy!\n");
}
vagrant@debian12:~$ ls | head # default first 10 lines
arguments
arguments.c
b
bakhsp02.sql
b.sh
```

```
ex231
hello
hello.c
hello.md
jdk-21.0.2
vagrant@debian12:~$ ls | head -n 3
arguments
arguments.c
b
vagrant@debian12:~$ ls | tail -n 2
week3
week4
vagrant@debian12:~$ cat fruit
apple
banana
apple
orange
banana
vagrant@debian12:~$ sort < fruit | uniq
apple
banana
orange
vagrant@debian12:~$ sort < fruit|uniq | wc -l
3
```

- `grep [-iv] EXPRESSION` **: read 1, prints only lines match the regular expression.**

  With `-i` it is case-insensitive; **with** `-v` **it only prints lines that do *not* match the expression.有排除用-v**

- `sed -e COMMAND` : read 0, transforms them according to command, write 1.

  Most common one is `s/SOURCE/DEST/` : SOURCE → DEST(changing)

- `wc [-l]` : for word count. - 一般放在最后面
  - Very **end** of a pipe is useful if you just want to know **how many results** a particular command or pipe produces, assuming the **results come one per line**.

All these commands actually take an optional extra filename as argument, in which case they read from this file as input. e.g.

```
vagrant@debian12:~$ head fruit
apple
banana
apple
orange
banana
vagrant@debian12:~$ cat fruit | head
apple
banana
apple
orange
banana
```

`wget` is one of two utilities for **downloading files**, the other being `curl`. Note that the option for output file name is a **capital O**, not a lowercase o or a zero.

`wget` `https://users.cs.duke.edu/~ola/ap/linuxwords` `-O words`

- The regular expression `X` would match an X anywhere in the word
- `^X` matches an X only at the start of the string.
- The expression `'j$'` matches a j only at the end of the string
- The expression `.` (period) matches a single character
- `'^...$'` for example would match all strings of the format *exactly three characters between start and end of string*.
- `'[aeiou]'` matches any string that contains one of the bracketed characters
- . Putting a `*` after something in a regular expression searches for *any number of repetitions of this, including 0.*
    - `'a*'` would find words with any number of the letter a, including 0.

grep

- `grep PATTERN FILE` **VS** `grep -w PATTERN FILE` **:** -w : only matches exactly word.
- `grep pattern file | wc -l` VS `grep -c PATTERN FILE` : second is better.-prevent lost data- more exactly.

Pipes Normal Exercise

Pipes Other Exercise

Additional materials: **华为大佬一周讲完的Linux三剑客grep/sed/awk** ；正则表达式专项训练