

Oscar Petersson, Johan Levinsson

TDDI41 Lab report

Högskoleingenjörsutbildning i datateknik, 180 hp

Laboration report - September 29, 2016

System Administration

TDDI41, Linköpings universitet

Assistant:

IDA

UML

3-2

64M for lab1, lab2 server has 128M

3-3

A “wrapper for terminal sessions”, or rather, an emulated terminal device. It allows the terminal session to be kept running (and resumed later) even when the terminal client itself is closed.

6-1

```
$ scp root@10.17.1.212:/etc/network/interfaces <wherever-we-want-to-put-it>
```

6-2

```
$ scp -r root@....:/etc/default <destination> foo
```

LXB

3-1

a)

- 1 Executable programs or shell commands
- 2 System calls (functions provided by the kernel)
- 3 Library calls (functions within program libraries)
- 4 Special files (usually found in /dev)
- 5 File formats and conventions eg /etc/passwd
- 6 Games
- 7 Miscellaneous (including macro packages and conventions), e.g. man(7), groff(7)
- 8 System administration commands (usually only for root)
- 9 Kernel routines

b) Section 1

c) Section 5

d) Section 8

4-1

a)

```
command argument argument
```

b)

```
command [runtime options goes here]...  
[optional input stream[optional output stream]]
```

c)

```
command [-available runtime flags w/o further arguments]  
[-S flag needs argument (filename suffix)] [optional output name]
```

d) Three possible invocation syntaxes:

```
command [runtime option] argument argument  
command [runtime option] [-u-flag argument] [-r-flag argument] argument  
command [runtime option] --reference-flag=argument argument
```

4-2

a) A brief briefing of what the entry is about, i.e. for an executable - what the executable' function is, and a description of flags etc.

b) Section 1

c) Under **SEE ALSO**

d) Section 1, usually a subsection

4-3

a) `man -a <search term>`

b) `man -k <keyword>`

4-4

a) It lists info about files in a given search path.

b) `-l`

c) Adds recursive directory search (lists subdirectories and their content as well).

5-1

a) `./ssh/`

`../bin/ssh/`

b) `../../bin/ssh/`

6-1

- a) # `chmod a+rw-x,go-w <file>`
- b) # `chmod g-w <file>`
- c) # `chmod ug+x foo`

7-1

- a) `rw-rw-rw-`
- b) `rw-rwx---`
- c) `rw-r-----`
- d) `r--r--r--`

7-2

`-R, --recursive`

7-3

Allows entering the directory and listing of its content.

7-4

(c) is the most suitable one, as it doesn't change possibly existing executive permissions for [ug]. (a) would remove any executive permissions on all files (and directories, making them inaccessible) for anyone, and (b) would add executive permissions for [ug] to all files and directories in the tree.

8-1

`chown user:group -R <path root>`

9-1,2,3

- 1) Changes current work directory to the parent directory.
- 2) Changes current work directory to the parent directory of the parent directory.
- 3) Current directory content in the format:
`filemode <> user group filesize(bytes) modification_time filename[entry type indicator]`

9-4

Character special file, e.g. pipes, ports, devices etc ...

9-5

rename file 'a' to 'b'. `mv -i` will interactively ask for permission to overwrite.

9-6

```
$ cp -a /dir1/*.* /dir2/
```

9-7

```
$ chown root:wheel secret && chmod 640 secret
```

10-2

<code>/bin/bash</code>	The bash executable
<code>/etc/profile</code>	The systemwide initialization file, executed for login shells
<code>/etc/bash.bashrc</code>	The systemwide per-interactive-shell startup file
<code>/etc/bash.bash.logout</code>	The systemwide login shell cleanup file, executed when a login shell exits
<code>~/.bash_profile</code>	The personal initialization file, executed for login shells
<code>~/.bashrc</code>	The individual per-interactive-shell startup file
<code>~/.bash_logout</code>	The individual login shell cleanup file, executed when a login shell exits
<code>~/.inputrc</code>	Individual readline initialization file

11-1

```
PATH=/bin:/sbin:/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/bin/site_perl:\n/usr/bin/vendor_perl:/usr/bin/core_perl
```

`$PATH` is a list of paths from where the shell will recognize executables. Executables in `$PATH` can be executed from any search path without writing the full path to the executable.

11-2

`$HOME` is the user "default" directory. Here the users document, user-specific configurations etc. are stored.

11-3

```
$ PATH='/data/kurs/adit/bin:/data/kurs/TDDI41/bin:${PATH}'
```

12-1

stdout:			stderr:		
	file1	console		file1	console
a	y	n	a	n	y
b	y	n	b	n	y
c	y	n	c	y	n

13-1

- a) List directory content, select only the lines which contains the ordered character set "doc" ignoring upper/lower case.
- b) Redirects stderr output of `command` to stdout which is then parsed for "fail" (ignoring u/l case)
- c) As b, but will also put the stdout output into The Great Void.

13-2

- a) `$ ls -aR ~ > /tmp/HOMEFILES`
- b) `# find / -perm -ow -type f 2>/dev/null=`

14-2

```
$ ping 127.0.0.1 >/dev/null&
kill $(ps -Ao pid=,comm --no-headers | grep ping | grep -o -E '[0-9]+')
```

(Or, just "ps", look for ping, send `kill <found PID>`), but that's a rather boring way, isn't it? ;)

14-3

```
$ kill -9 <pid>
```

This line sends **SIGKILL** to the process with given PID. This can not be caught or ignored by the process.

```
$ kill -9 -1
```

This line sends **SIGKILL** to all processes with a PID larger than 1, i.e. every process apart from the init process started by the boot manager.

14-4

```
$ pkill ping
```

16-1,2,3,4

BOF: g

EOF: G

forward search: /option

backward search: ?option

16-5

a) For security reasons, `X11Forwarding` is disabled by default, this goes for Debian's `openssh-server` package as well.

b) `xauth` at a bare minimum, plus of course any related packages for whatever functionality desired.

17-1 (optional)

```
# sed -i "s|bin/tcsh|bin/sh|g" < /etc/passwd > foo
```

17-2 (optional)

```
$ paste <(awk -F ":" '{print $2}' passwd) <(awk -F ":" '{print $3}' shadow)
```

18-1

Prints the last 10 lines of `/var/log/syslog` and keeps parsing.

18-2

```
$ cat /var/log/syslog | grep cron
```

Part 5 Errata:

The default configuration for most contemporary distributions is `systemd`, regardless of what one thinks of that. Sure, `sysvinit` might be a reasonable (at least you can reason whether it is or not ...) choice regarding this course, but that does not change the fact that the major distributions likely to be found in a professional environment (Debian, Fedora, SUSE, Red Hat, buntu, Mint)- except for legacy systems - are not running `sysvinit`.

19-1

bootlogs, syslog, ssh, cron, rnmologin, stopbootlogs

19-2

```
$ systemctl restart sshd.service
-bash: systemctl: command not found
```

Oh right, old habits die hard ... :p

```
$ /etc/init.d/ssh restart
```

20-1

see 12-1 b/c

20-2

```
$ kill $(ps -Ao pid,comm --no-headers | grep ping | grep -o -E '[0-9]+')
```

20-3

```
# ~/.aliases
gzless() {
    gunzip -c $1 | less
}
```

20-4

a)

```
$ file=$(cat commandfile) && \
    (sleep 3; echo PASSWD; sleep 1; echo "$file"; sleep 1) |\
    socat - EXEC:'ssh -l USER remote.ip.addr',pty,setsid,ctty
```

b)

20-5

```
#!/bin/bash
```

```
for OLD in `find $1 -depth`
do
    NEW=`dirname "${OLD}"`/`basename "${OLD}" | tr ' [A-Z ]' '[a-z ]`
    if [ "${OLD}" != "${NEW}" ]
    then
        [ ! -e "${NEW}" ] && mv -T "${OLD}" "${NEW}" \
        || echo "Could not rename ${OLD}. ${NEW} already exists."
    fi
done
```


20-6

see 17-2

20-7

```
# find ./ -type f | xargs grep "$oldip" 2>/dev/null ...where $oldip is the IP-address
we are looking for, of course.
```

20-8

```
# find /etc -type f -exec sed -i "s|$OLD|$NEW|g" {} \;
```

20-9

Avoiding password login would be done through pubkey-based access:

```
#/etc/ssh/sshd_config
PasswordAuthentication no
PermitEmptyPasswords no
RSAAuthentication yes
DSAAuthentication yes
HostKey <key path>
[...]
#EOF
```

Generating and importing keys ...

Once sorted (phew), and supposing we have the computers defined in `/etc/hosts`, we'll simply run

```
$ for client in `seq 0001 2000`
do
    ssh user@$client <<FOOEXECUTETHIS
    w
    FOOEXECUTETHIS
done
```

21-1

21-2

The code starts out with directing two file descriptors, one for input and one for output (1.2). We look for .bak-files throughout the file system and print their path on a line each (1.3). We read these lines (1.5), ask (1.6) the user to confirm that we should log the entry to `/tmp/RECORD` (1.2). If (1.8) the user answers (1.7) "y", we log the entry (1.2).

21-3

```
#!/bin/bash
exec 23<&0 24>&1 <<EOF
'find . -name "*.bak" -print'
EOF
while read BAK; do
    FILE=$(echo "$BAK" | sed 's|\.bak$||g')
    if [ ! -e "$FILE" ] || [ "${BAK}" -nt "${FILE}" ]; then
        echo -n "$FILE not found or older than $BAK. Replace $FILE? Y/N"
        read ans <&23
        if [ "$ans" = "y" ] || [ "$ans" = "Y" ]; then
            mv -T "$BAK" "$FILE"
        fi
    fi
done
```

APT

Errata: "[...]dpkg, is one of the most powerful"^{citation needed}

1-1

- a) `# apt-get install <package>`
- b) `# apt-get remove|purge <package>`
- c) `update` gets the latest index files from the repositories.
- d) It upgrades all explicitly installed packages and (hopefully) their dependencies.

1-2

- a) Prints a list of installed packages, unless a search pattern is given, in which case not installed in the indexes also are included (well, the ones that matches the search pattern).
- b) Removes the package and its config files.
- c) Lists files installed by `bind9`.

6

- 1) `/ <word>`
- 2) `l <word>`
- 3)
 - `# aptitude install <package>`
 - `# aptitude remove <package>`
 - (`'+'` and `'-'` keys)
- 4) `# aptitude -i`
(`'g'`-key)
- 5) The package has broken dependencies.

SCT

1-1

`$@` looks at each argument individually, whereas `@*` regards them as a collection as shown by the example below:

```
#foo.sh
#!/bin/bash
printf "%s\n" "$@"
printf "%s\n" "$*"
#EOF

$ ./foo.sh meep moop
meep
moop
meep moop
```

2-1

`$@`

will make another evaluation of the parameter which can yield unwanted results with reserved characters. For instance, if fed `'ls -F'` when an executable is present in the path (annotated in the output with `'*'`, e.g. `foo*`), it will evaluate the expression (`foo*` becomes *"all files beginning with"foo"*) which might result in duplicate arguments.

2-2

Yes, and no. `$*` could work, but we would still run into the issues mentioned in 2-1 with evaluation of the arguments provided. `$*` would not work however, unless it is only one file provided as argument.

3-1

```
#!/bin/bash
i=10
while [ $i -ne 0 ]
do
```

```
    echo "$i"
    i=$((i - 1))
done
```

5-1

Evaluates `$line`, string position 0, 1 character(s).

5-2

`:` == true, which is needed as conditional blocks can't be empty.

5-3

`$(command)`

5-4

Arithmetic expansion. Allows evaluation of arithmetic expressions and substitution of the result, allowing nesting for instance.

5-5

```
$ ls -a | grep "^\. " | wc -l
```

6-1

a) A backslash is used as an escape character in bash, and in this case it escapes the newline, allowing the script to be written on several (two here) lines while being evaluated as if it was a single line.

b) Earlier, we saw that command scope within `()` are executed in a new sub-shell. If replaced by `{}` the scope is executed in the current shell context.

6-2

`local` creates a temporary object that is local to the current call frame, in this case a variable `$mailserver`.

6-3

To demonstrate `sleep`?

7-1, 8-1

See separate files.

NET

1-1

Ping is used to check and time connectivity between two nodes (a). It uses the ICMP protocol to send ECHO_REQUEST and ECHO_RESPONSE datagrams (c).

b) `ping -n <address>`

d) By `ping` alone, we need to know that the host, and the routers leading there, allows ICMP datagrams. But, assuming they do, `ping 10.17.1.1` should do the trick. We will then get a response time.

1-2

a) There are often multiple possible paths between nodes, and `traceroute` tries to list a possible path between two nodes (client, target). Over Internet, we can often run into incomplete traces though as ICMP datagrams and not-explicitly-used UDP ports are often blocked by firewalls. Therefore, http-packets to ports known to be in use (i.e. 80 for a web server, 25 for a mail server, etc.) can be needed to give a proper path.

b) `traceroute` sends IP packets with an increasing ttl, and listening for responses claiming it to have exceeded its ttl. This keeps on going until we get an "unreachable" or TCP reset response, at which time we will have reached the target.

c) `-n`

d)

`traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets`

```
1  c-210-4.eduroam.liu.se (130.236.210.4)  14.118 ms  17.596 ms  18.731 ms
2  130.236.7.61 (130.236.7.61)  19.058 ms  19.075 ms  20.394 ms
3  130.236.9.198 (130.236.9.198)  22.094 ms  22.349 ms  22.350 ms
4  liu-br2.sunet.se (193.11.0.21)  22.340 ms  22.328 ms  24.651 ms
5  m1tug-xe-7-3-3.sunet.se (130.242.85.173)  29.593 ms  31.280 ms  31.282 ms
6  se-tug.nordu.net (109.105.102.17)  31.273 ms  9.483 ms  4.631 ms
7  google-gw.nordu.net (109.105.98.6)  4.609 ms  9.961 ms  9.962 ms
8  216.239.40.27 (216.239.40.27)  11.039 ms  11.038 ms  11.028 ms
9  209.85.242.47 (209.85.242.47)  11.030 ms  11.023 ms  72.14.234.85 (72.14.234.85)  11.017
10 google-public-dns-a.google.com (8.8.8.8)  10.969 ms  10.946 ms  10.939 ms
```

Here we see an example of a 10-hop path from 130.236.210.157 to a Google DNS server (8.8.8.8) with 60 byte packets (IPv4). By default, we send three probe packets per hop and therefore we

end up with three response times for each hop. The times are between current and previous node. We also see the nodes' name addresses (where applicable) and IP addresses.

1-3

```
a) ifconfig eth0 130.236.189.14/24 broadcast 130.236.189.255
ip addr add 130.236.189.14/24 broadcast 130.236.189.255 dev eth0
```

1-4

a) `sysctl` in Linux is used to examining and modify kernel parameters at runtime" (SYSCTL(8)). This is done through files in `procfs`' virtual process file system (`/proc/sys`).

b) If we want to load certain values at boot time, we have to know what init-system we use. For instance, in `systemd`, we'll put the config files in `/etc/system.d/`, whereas in `sysvinit` we'll use `/etc/sysctl.conf`.

c) `net.ipv<4|6>.conf.<device|all>.forwarding`

1-5

FQDN: b4.sysinst.ida.liu.se

Internal IP range: 130.236.178.152/29 (i.e. 130.236.178.(152-159, brd 159, hosts 153-158))

Router external IP: 130.236.178.17

1-6

Routing is finding out where the destination of the packet is, forwarding is just passing the packet on, not knowing whether it would reach the destination or not.