Cal State Fullerton

CPSC 254

Software Development With Open Source Systems - Linux Utilities

Instructor: Tejaas Mukunda Reddy

Linux like Unix comes equipped with a number of useful utilities

grep

- https://www.gnu.org/software/grep/manual/grep.html
- grep searches input files for lines containing a match to a given pattern list. When it finds a match in a line, it copies the line to standard output (by default) or produces whatever other sort of output you have requested with options.
- Though grep expects to do the matching on text, it has no limits on input line length other than available memory, and it can match arbitrary characters within a line. If the final byte of an input file is not a newline, grep silently supplies one. Since newline is also a separator for the list of patterns, there is no way to match newline characters in a text.
- The general synopsis of the grep command line is
- grep options pattern input_file_names
- There can be zero or more options. pattern will only be seen as such (and not as an input_file_name) if it wasn't already specified within options (by using the '-e pattern' or '-f file' options). There can be zero or more input_file_names.
- https://www.linode.com/docs/tools-reference/tools/how-to-grep-for-text-in-files/

grep

- While straightforward pattern matching is sufficient for some filtering tasks, the true power of grep lies in its ability to use regular expressions for complex pattern matching. Most characters in regular expressions match with input data literally; however, there are some sequences that carry special significance:

Symbol	Result
.	Matches any character.
*	Matches zero or more instances of the preceding character.
+	Matches one or more instances of the preceding character.
[]	Matches any of the characters within the brackets.
()	Creates a sub-expression that can be combined to make more complicated expressions.
|	**OR** operator; (www|ftp) matches either “www” or “ftp”.
^	Matches the beginning of a line.
$	Matches the end of the line.
\	Escapes the following character. Since . matches any character, to match a literal period you would need to use \..

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

grep

- One popular use of grep is to extract useful information from system logs:

```
# grep -Eoc "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}.* 200"  /srv/www/example.com/logs/access.log
```

- In this command, grep filters an Apache access log for all lines that begin with an IP address, followed by a number of characters, a space and then the characters 200 (where 200 represents a successful HTTP connection). The -c option outputs only a count of the number of matches. To get the output of the IP address of the visitor and the path of the requested file for successful requests, omit the -c flag:

```
# grep -Eo "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}.* 200"  /srv/www/example.com/logs/access.log
```

- The curly brackets specify the number of instances of the pattern. {1,3} requires that the previous character occur at least once, but no more than three times. The character class [0-9] will match against one or more numeric digits. You can also generate similar output but report on unsuccessful attempts to access content:

```
# grep -Eo "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}.* 404"  /srv/www/example.com/logs/access.log
```

grep

- The following command generates a list of all IP addresses that have attempted to connect to your web server. Using the -o option, only the matching strings are sent to standard output. This output is filtered through the utility uniq with the pipe operator (|) to filter out duplicate entries:

```
# grep -Eo "^[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" /srv/www/example.com/logs/access.log | uniq
```

- The next example uses an alternative pattern for matching an IP address in a different log. The following command searches the most recent /var/log/auth.log file for invalid login attempts:

```
# grep -Eo "Invalid user.*([0-9]{1,3}\.){3}[0-9]{1,3}" /var/log/auth.log
```

- You can split the above command into two layers to output a list of IP addresses with failed login attempts to your system:

```
# grep "Invalid user" /var/log/auth.log | grep -Eo "([0-9]{1,3}\.){3}[0-9]{1,3}" | uniq
```

- grep can filter the output of commands such as tail -F to provide real-time monitoring of specific log events:

```
# tail ~/.procmail/procmail.log -F | grep "Subject"
```

- In this case, tail follows the ~/procmail/procmail.log file. This output is passed to grep, which filters the stream and prints only lines that contain the string “Subject”.

ping

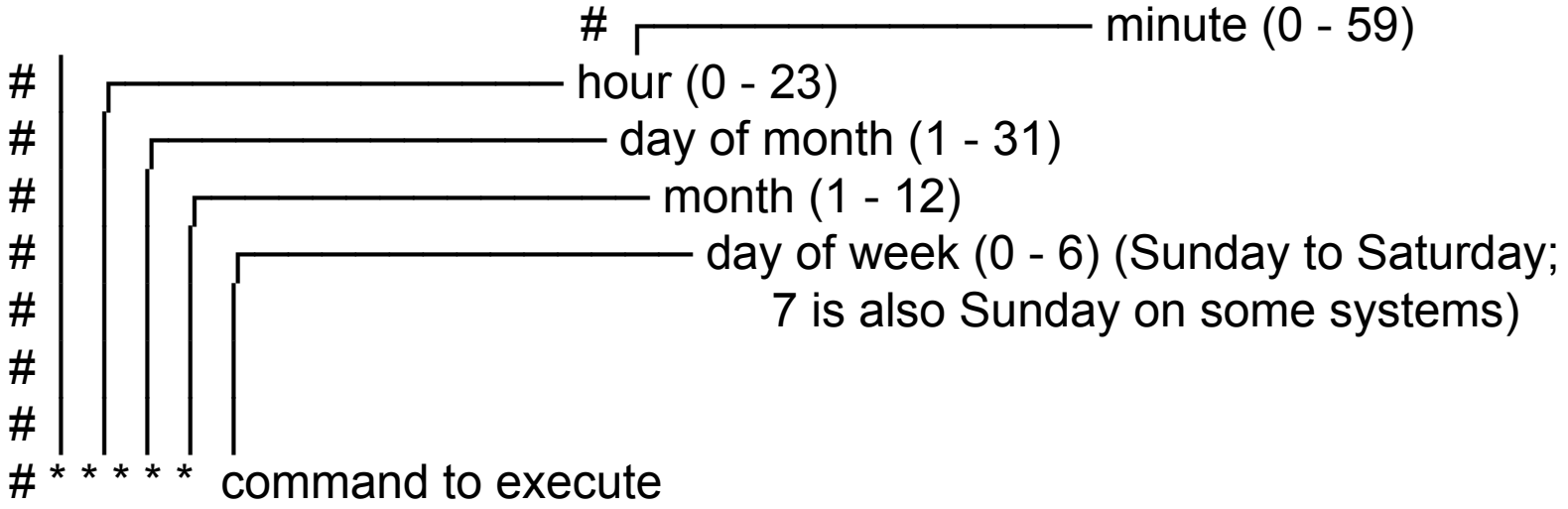
- The ping command is one of the most often used networking utilities for troubleshooting network problems.
- https://www.lifewire.com/ping-command-2618099
- The ping command is a Command Prompt command used to test the ability of the source computer to reach a specified destination computer. The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device.
- The ping command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waiting for a response.
- How many of those responses are returned, and how long it takes for them to return, are the two major pieces of information that the ping command provides.
- For example, you might find that there are no responses when pinging a network printer, only to find out that the printer is offline and its cable needs replaced. Or maybe you need to ping a router to verify that your computer can connect to it, to eliminate it as a possible cause for a networking issue.
- The ping command is usually used as a simple way to verify that a computer can communicate over the network with another computer or network device. The ping command operates by sending Internet Control Message Protocol (ICMP) Echo Request messages to the destination computer and waiting for a response.

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

crontab

- https://en.wikipedia.org/wiki/Cron
- The software utility cron is a time-based job scheduler in Unix-like computer operating systems. People who set up and maintain software environments use cron to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or intervals. It typically automates system maintenance or administration—though its general-purpose nature makes it useful for things like downloading files from the Internet and downloading email at regular intervals.
- Cron is driven by a crontab (cron table) file, a configuration file that specifies shell commands to run periodically on a given schedule. The crontab files are stored where the lists of jobs and other instructions to the cron daemon are kept. Users can have their own individual crontab files and often there is a system-wide crontab file (usually in /etc or a subdirectory of /etc) that only system administrators can edit.
- Each line of a crontab file represents a job, and looks like this:



Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

crontab

- The syntax of each line expects a cron expression made of five fields, followed by a shell command to execute.
- While normally the job is executed when the time/date specification fields all match the current time and date, there is one exception: if both "day of month" (field 3) and "day of week" (field 5) are restricted (not "*"), then one or both must match the current day.[3]
- For example, the following clears the Apache error log at one minute past midnight (00:01) every day, assuming that the default shell for the cron user is Bourne shell compliant:
- 1 0 * * * printf "" > /var/log/apache/error_log
- This example runs a shell program called export_dump.sh at 23:45 (11:45 PM) every Saturday.
- 45 23 * * 6 /home/oracle/scripts/export_dump.sh
- The configuration file for a user can be edited by calling crontab -e regardless of where the actual implementation stores this file
- Some cron implementations, such as the popular 4th BSD edition written by Paul Vixie and included in many Linux distributions, add a sixth field: an account username that runs the specified job (subject to user existence and permissions). This is allowed only in the system crontabs—not in others, which are each assigned to a single user to configure. The sixth field is alternatively sometimes used for year instead of an account username—the nncron daemon for Windows does this.
- Nonstandard predefined scheduling definitions[edit]

crontab

- Some cron implementations[4] support the following non-standard macros:

Entry	Description	Equivalent to
@yearly (or @annually)	Run once a year at midnight of 1 January	0 0 1 1 *
@monthly	Run once a month at midnight of the first day of the month	0 0 1 * *
@weekly	Run once a week at midnight on Sunday morning	0 0 * * 0
@daily	Run once a day at midnight	0 0 * * *
@hourly	Run once an hour at the beginning of the hour	0 * * * *
@reboot	Run at startup	N/A

crontab

- @reboot configures a job to run once when the daemon is started. Since cron is typically never restarted, this typically corresponds to the machine being booted. This behavior is enforced in some variations of cron, such as that provided in Debian,[5] so that simply restarting the daemon does not re-run @reboot jobs.
- @reboot can be useful if there is a need to start up a server or daemon under a particular user, and the user does not have access to configure init to start the program.
- cron permissions[edit]
- These two files play an important role:
- /etc/cron.allow - If this file exists, it must contain your username for you to use cron jobs.
- /etc/cron.deny - If the cron.allow file does not exist but the /etc/cron.deny file does exist then, to use cron jobs, you must not be listed in the /etc/cron.deny file.
- Note that if neither of these files exist then, depending on site-dependent configuration parameters, either only the super user can use cron jobs, or all users can use cron jobs.

crontab

- CRON expression. A CRON expression is a string comprising five or six fields separated by white space[10] that represents a set of times, normally as a schedule to execute some routine.
- Comments begin with a comment mark #, and must be on a line by themselves.

Field	Required	Allowed values	Allowed special characters	Remarks
Minutes	Yes	0–59	* , -	
Hours	Yes	0–23	* , -	
Day of month	Yes	1–31	* , - ? L W	? L W only in some implementations
Month	Yes	1–12 or JAN–DEC	* , -	
Day of week	Yes	0–6 or SUN–SAT	* , - ? L #	? L W only in some implementations
Year	No	1970–2099	* , -	This field is not supported in standard/default implementations.

crontab

- The month and weekday abbreviations are not case-sensitive.
- In the particular case of the system crontab file (/etc/crontab), a user field inserts itself before the command. It is generally set to 'root'.
- In some uses of the CRON format there is also a seconds field at the beginning of the pattern. In that case, the CRON expression is a string comprising 6 or 7 fields.[11]
- Comma (,)
  - Commas are used to separate items of a list. For example, using "MON,WED,FRI" in the 5th field (day of week) means Mondays, Wednesdays and Fridays.
- Hyphen (-)
  - Hyphens define ranges. For example, 2000–2010 indicates every year between 2000 and 2010, inclusive.
- Percent (%)
  - Percent-signs (%) in the command, unless escaped with backslash (\), are changed into newline characters, and all data after the first % are sent to the command as standard input.

mount

- https://www.computerhope.com/unix/umount.htm
- The mount command mounts a storage device or filesystem, making it accessible and attaching it to an existing directory structure.
- The umount command "unmounts" a mounted filesystem, informing the system to complete any pending read or write operations, and safely detach it.
- All files accessible in Unix, or a Unix-style system such as Linux, are arranged in one big tree: the file hierarchy, rooted at /. These files can be spread out over several devices. The mount command attaches a filesystem, located on some device or other, to the file tree. Conversely, the umount command will detach it again.
- The standard form of the mount command is: # mount -t type device dir
- This tells the kernel to attach the filesystem found on device (which is of type type) at the directory dir. The previous contents (if any), owner, and mode of dir become invisible, and as long as this filesystem remains mounted, the pathname dir refers to the root of the filesystem on device.
- The file /etc/fstab may contain lines describing what devices are usually mounted where, using which options. The command #mount -a [-t type] [-O optlist]
- causes all filesystems mentioned in fstab (of the proper type and/or having or not having the proper options) to be mounted as indicated, except for those whose line contains the noauto keyword. This command would typically be included in a boot script. Adding the -F option will make mount fork, so that the filesystems are mounted simultaneously.
- When mounting a filesystem mentioned in fstab or mtab, it suffices to give only the device, or only the mount point. The programs mount and umount maintain a list of currently mounted filesystems in the file /etc/mtab. If no arguments are given to mount, this list is printed.
- In general, fstab is no longer necessary for auto mounting. Modern Linux systems do this like Windows does.

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

alias

- http://www.linfo.org/alias.html
- The alias command makes it possible to launch any command or group of commands (inclusive of any options, arguments and redirection) by entering a pre-set string (i.e., sequence of characters).
- That is, it allows a user to create simple names or abbreviations (even consisting of just a single character) for commands regardless of how complex the original commands are and then use them in the same way that ordinary commands are used.
- A command is an instruction given by a user to tell a computer to do something. Commands are generally issued by typing them in at the command line (i.e., an all-text user interface) and then pressing the ENTER key, which passes them to the shell. A shell is a program that provides the traditional, text-only user interface for a Unix-like operating systems. Its primary function is to read commands and then execute (i.e., run) them.
- The alias command is built into a number of shells including ash, bash (the default shell on most Linux systems), csh and ksh. It is one of several ways to customize the shell (another is setting environmental variables). Aliases are recognized only by the shell in which they are created, and they apply only for the user that creates them, unless that user is the root (i.e., administrative) user, which can create aliases for any user.
- Listing and Creating Aliases
- The general syntax for the alias command varies somewhat according to the shell. In the case of the bash shell it is
- alias [-p] [name="value"]
- When used with no arguments and with or without the -p option, alias provides a list of aliases that are in effect for the current user, i.e.,

alias

- Uses for alias
  - Reducing the amount of typing that is necessary for commands or groups of commands that are long and/or tedious to type. These commands could include opening a file that is frequently used for studying or editing.
  - A second type of use for aliases is specifying the default version of a program that exists in several versions on a system or specifying default options for a command. For example, the following alias would have the ls command always list all items in a directory rather than just the non-hidden ones:
    - # alias ls="ls -a"
  - A third use of aliases is correcting common misspellings of commands. For example, if a user has a habit of accidentally typing pdw instead of pwd, the following command will create an alias so that either spelling will work:
    - # alias pdw="pwd"
  - A fourth use of aliases is increasing the safety of the system by making commands interactive. This forces the user to confirm that it is desired to perform a specific action and thereby reduces the risk from accidental or impulsive abuse of powerful commands. For example, the rmcommand, which can remove files and directories and make them virtually unrecoverable, can be made interactive as follows:
    - # alias rm="rm -i"
  - Likewise, the risks associated with the cp command, which is used to copy the contents of one file to another file, can also be reduced by making it interactive by default. If the name for the file to be written to does not exist in the specified directory (by default the current directory), it will be created, but if it already exists, its contents will be overwritten. Thus, creating the following alias will reduce the chances of an unintended overwriting:
    - # alias cp="cp -i"

alias

- Another use of aliases is standardizing the name of a command across multiple operating systems. For example, different versions of Linux or other operating systems contain different versions of the vi text editor (i.e., vi or its clones vim, nvi, elvis, etc.), but issuing the vi command on any of these divergent systems will generally launch the particular vi clone that is resident on that system (assuming that the appropriate alias has been created). For instance, Red Hat Linux installs vim by default, but issuing the command vi launches vim because the alias alias vi="vim" is also installed by default for all users for the bash, csh and tcsh shells. Of course, the command vim can also be used, but vi is easier for most people to remember.
- For people accustomed to MS-DOS commands, the following aliases can be defined so that a Unix-like operating system appears to behave more like MS-DOS:
    - alias dir="ls"
    - alias copy="cp"
    - alias rename="mv"
    - alias md="mkdir"
    - alias rd="rmdir"
    - alias del="rm -i"
- However, some experienced users of Unix-like systems contend that this may not be a good idea and that it might just make Linux seem more confusing, rather than simpler. Instead, they advocate having Linux users become accustomed to the UNIX terminology right from the start

tar

- https://www.howtogeek.com/248780/how-to-compress-and-extract-files-using-the-tar-command-on-linux/
- The tar command on Linux is often used to create .tar.gz or .tgz archive files, also called “tarballs.” This command has a large number of options, but you just need to remember a few letters to quickly create archives with tar. The tar command can extract the resulting archives, too.
- The GNU tar command included with Linux distributions has integrated compression. It can create a .tar archive and then compress it with gzip or bzip2 compression in a single command. That’s why the resulting file is a .tar.gz file or .tar.bz2 file.
- Use the following command to compress an entire directory or a single file on Linux. It’ll also compress every other directory inside a directory you specify–in other words, it works recursively.
    - # tar -czvf name-of-archive.tar.gz /path/to/directory-or-file
- Here’s what those switches actually mean:
    - -c: Create an archive.
    - -z: Compress the archive with gzip.
    - -v: Display progress in the terminal while creating the archive, also known as “verbose” mode. The v is always optional in these commands
    - -f: Allows you to specify the filename of the archive.
- Use bzip2 Compression Instead. While gzip compression is most frequently used to create .tar.gz or .tgz files, tar also supports bzip2 compression. This allows you to create bzip2-compressed files, often named .tar.bz2, .tar.bz, or .tbz files. To do so, just replace the -z for gzip in the commands here with a -j for bzip2.
- Gzip is faster, but it generally compresses a bit less, so you get a somewhat larger file. Bzip2 is slower, but it compresses a bit more, so you get a somewhat smaller file. Gzip is also more common, with some stripped-down Linux systems including gzip support by default, but not bzip2 support. In general, though, gzip and bzip2 are practically the same thing and both will work similarly.

sudo

- https://www.linux.com/blog/how-use-sudo-and-su-commands-linux-introduction
- Introduction to Linux command 'sudo'
- In Ubuntu Linux there is not root account configured by default. If users want root account password then they can manually set it up oo can use 'sudo'. As we all know, Linux in many ways protects users' computer being used for bad purposes by some nasty people around us. Using sudo is one of those good ways. Whenever a user tries to install, remove and change any piece of software, the user has to have the root privileges to perform such tasks. sudo, linux command is used to give such permissions to any particular command that a user wants to execute. sudo requires the user to enter user password to give system based permissions. For example user wants to update the operating system by passing command
    - # apt-get update – You can get an error this error is due to not having root privileges to the user 'sandy'. The root privileges can be required by passing sudo at the very beginning, like
    - # sudo apt-get update

ssh

- https://www.ssh.com/ssh/command/
- Practically every Unix and Linux system includes the ssh command. This command is used to start the SSH client program that enables secure connection to the SSH server on a remote machine. The ssh command is used from logging into the remote machine, transferring files between the two machines, and for executing commands on the remote machine.
- SSH COMMAND IN LINUX
- The ssh command provides a secure encrypted connection between two hosts over an insecure network. This connection can also be used for terminal access, file transfers, and for tunneling other applications. Graphical X11 applications can also be run securely over SSH from a remote location.
- OTHER SSH COMMANDS
- There are other SSH commands besides the client ssh. Each has its own page.
    - ssh-keygen - creates a key pair for public key authentication
    - ssh-copy-id - configures a public key as authorized on a server
    - ssh-agent - agent to hold private key for single sign-on
    - ssh-add - tool to add a key to the agent
    - scp - file transfer client with RCP-like command interface
    - sftp - file transfer client with FTP-like command interface
    - sshd - OpenSSH server

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

ssh

- USING THE LINUX CLIENT
- Linux typically uses the OpenSSH client. The ssh command to log into a remote machine is very simple. To log in to a remote computer called sample.ssh.com, type the following command at a shell prompt:
  - # ssh sample.ssh.com
- If this is the first time you use ssh to connect to this remote machine, you will see a message like:
- The authenticity of host 'sample.ssh.com' cannot be established.
- DSA key fingerprint is 04:48:30:31:b0:f3:5a:9b:01:9d:b3:a7:38:e2:b1:0c.
- Are you sure you want to continue connecting (yes/no)?
- Type yes to continue. This will add the server to your list of known hosts (~/.ssh/known_hosts) as seen in the following message:
- Warning: Permanently added 'sample.ssh.com' (DSA) to the list of known hosts.
- Each server has a host key, and the above question related to verifying and saving the host key, so that next time you connect to the server, it can verify that it actually is the same server.
- Once the server connection has been established, the user is authenticated. Typically, it asks for a password. For some servers, you may be required to type in a one-time password generated by a special hardware token.
- Once authentication has been accepted, you will be at the shell prompt for the remote machine.

ssh

- SSH CLIENT CONFIGURATION FILE
- The ssh command reads its configuration from the SSH client configuration file ~/.ssh/config. For more information, see the page on SSH client configuration file.
- CONFIGURING PUBLIC KEY AUTHENTICATION
- To configure passwordless public key authentication, you may want to create an SSH key and set up an authorized_keys file. See the pages on ssh-keygen and ssh-copy-id for more information.
- CONFIGURING PORT FORWARDING
- Command-line options can be used to set up port forwarding. Local fowarding means that a local port (at the client computer) is tunneled to an IP address and port from the server. Remote forwarding means that a remote port (at the server computer) is forwarded to a given IP address and port from the client machine. See the page on configuring port forwarding on how to configure them.
- OpenSSH also supports forwarding Unix domain sockets and IP packets from a tunnel device to establish a VPN (Virtual Private Network).

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

ssh

- SSH COMMAND LINE OPTIONS
- Some of the most important command-line options for the OpenSSH client are:
  - -1 Use protocol version 1 only, -2 Use protocol version 2 only, -4 Use IPv4 addresses only, -6 Use IPv6 addresses only.
  - -A Enable forwarding of the authentication agent connection, -a Disable forwarding of the authentication agent connection, -C Use data compression
  - -c cipher_spec Selects the cipher specification for encrypting the session.
  - -D [bind_address:]port Dynamic application-level port forwarding. This allocates a socket to listen to port on the local side. When a connection is made to this port, the connection is forwarded over the secure channel, and the application protocol is then used to determine where to connect to from the remote machine.
  - -E log_file Append debug logs to log_file instead of standard error.
  - -F configfile Specifies a per-user configuration file. The default for the per-user configuration file is ~/.ssh/config.
  - -g Allows remote hosts to connect to local forwarded ports, -i identity_file A file from which the identity key (private key) for public key authentication is read.
  - -J [user@]host[:port] Connect to the target host by first making a ssh connection to the pjump host[(/iam/jump-host) and then establishing a TCP forwarding to the ultimate destination from there.
  - -l login_name Specifies the user to log in as on the remote machine, -p port Port to connect to on the remote host.
  - -q Quiet mode, -V Display the version number.
  - -v Verbose mode., -X Enables X11 forwarding.

ssh

- A LITTLE HISTORY
- SSH replaced several older commands and protocols in Unix and Linux the 1990s. They include telnet, rlogin, and rsh.
- SSH runs at TCP/IP port 22. This is right between ftp and telnet, which are 20 years older. Read the story of how SSH got port 22.
- The following video summarizes how and why SSH was originally developed.
- https://www.youtube.com/watch?v=OHBdKM7s5V4

Prepared By: Tejaas Mukunda Reddy, Shivansh Vijay Nathan

Reference Prof. David Heckathorn

iptables

- https://www.howtogeek.com/177621/the-beginners-guide-to-iptables-the-linux-firewall/
- iptables is a command-line firewall utility that uses policy chains to allow or block traffic. When a connection tries to establish itself on your system, iptables looks for a rule in its list to match it to. If it doesn't find one, it resorts to the default action.
- iptables almost always comes pre-installed on any Linux distribution. To update/install it, just retrieve the iptables package:
- sudo apt-get install iptables
- There are GUI alternatives to iptables like Firestarter, but iptables isn't really that hard once you have a few commands down. You want to be extremely careful when configuring iptables rules, particularly if you're SSH'd into a server, because one wrong command can permanently lock you out until it's manually fixed at the physical machine.
- Types of Chains
- iptables uses three different chains: input, forward, and output.
- Input – This chain is used to control the behavior for incoming connections. For example, if a user attempts to SSH into your PC/server, iptables will attempt to match the IP address and port to a rule in the input chain.
- Forward – This chain is used for incoming connections that aren't actually being delivered locally. Think of a router – data is always being sent to it but rarely actually destined for the router itself; the data is just forwarded to its target. Unless you're doing some kind of routing, NATing, or something else on your system that requires forwarding, you won't even use this chain.

iptables

- There's one sure-fire way to check whether or not your system uses/needs the forward chain.
- # iptables -L -v
- The screenshot above is of a server that's been running for a few weeks and has no restrictions on incoming or outgoing connections. As you can see, the input chain has processed 11GB of packets and the output chain has processed 17GB. The forward chain, on the other hand, has not needed to process a single packet. This is because the server isn't doing any kind of forwarding or being used as a pass-through device.
- Output – This chain is used for outgoing connections. For example, if you try to ping howtogeek.com, iptables will check its output chain to see what the rules are regarding ping and howtogeek.com before making a decision to allow or deny the connection attempt.
- The caveat
- Even though pinging an external host seems like something that would only need to traverse the output chain, keep in mind that to return the data, the input chain will be used as well. When using iptables to lock down your system, remember that a lot of protocols will require two-way communication, so both the input and output chains will need to be configured properly. SSH is a common protocol that people forget to allow on both chains.
- History
- https://mediatemple.net/community/products/dv/204404624/using-the-history-command
- In Linux, there is a very useful command to show you all of the last commands that have been recently used. The command is simply called history, but can also be accessed by looking at your .bash_history in your home folder. By default, the history command will show you the last five hundred commands you have entered.
- This is especially useful during a live forensics investigation. passwd