Linux Basics

Content

- Video Session
  - How Linux is Built
  - What a Year for Linux
- Linux Introduction
  - History
  - Linux kernel Key features
  - Linux License
  - Everything is a File
  - Linux File System Structure
- Shell and File Handling
- Command Documentation
- Users and Permissions
- Standard I/O, redirections, pipes
- Looking for files
- Compressing and archiving
- Miscellaneous

Linux Basics

Linux Introduction

Linux History

- The Linux kernel is one component of a system, which also requires libraries and applications to provide features to end users.
- The Linux kernel was created as a hobby in 1991 by a Finnish student, **Linus Torvalds.**
- Linux quickly started to be used as the kernel for free software operating systems
- Linus Torvalds has been able to create a large and dynamic developer and user community around Linux.
- Nowadays, more than one thousand people contribute to each kernel release, individuals or companies big and small.

Linux Kernel Features

- **Portability** and hardware support. Runs on most architectures.
- **Scalability.** Can run on super computers as well as on tiny devices (4 MB of RAM is enough).
- **Compliance to standards** and interoperability.
- **Exhaustive networking** support.
- **Security.** It can't hide its flaws. Its code is reviewed by many experts.
- **Stability and reliability.**
- **Modularity.** Can include only what a system needs even at run time.
- **Easy to program.** You can learn from existing code. Many useful resources on the net.

Linux License

- The whole Linux sources are Free Software released under the GNU **General Public License version 2 (GPL v2)**.
- For the Linux kernel, this basically implies that:
  - When you receive or buy a device with Linux on it, you should receive the Linux sources, with the right to study, modify and redistribute them.
  - When you produce Linux based devices, you must release the sources to the recipient, with the same rights, with no restriction.

In Linux, Everything is a file

- Regular files
- Directories
  Directories are just files listing a set of files
- Symbolic links
  Files referring to the name of another file
- Devices and peripherals
  Read and write from devices as with regular files
- Pipes
  Used to cascade programs
  `cat *.log | grep error`
- Sockets
  Inter process communication

File names

File name features since the beginning of Unix

- Case sensitive
- No obvious length limit
- Can contain any character (including whitespace, except /).
  File types stored in the file (“magic numbers”).
  File name extensions not needed and not interpreted. Just used for user convenience.
- File name examples:
  README .bashrc Windows Buglist
  index.htm index.html index.html.old

File paths

A *path* is a sequence of nested directories with a file or directory at the end, separated by the / character

- **Relative path:** documents/fun/microsoft_jokes.html
  Relative to the current directory
- **Absolute path:** /home/bill/bugs/crash9402031614568
- / : *root directory*.
  Start of absolute paths for all files on the system (even for files on removable devices or network shared).

Linux filesystem structure (1)

Name	Description
/	Root directory
/bin/	Basic, essential system commands
/boot/	Kernel images, initrd and configuration files
/sbin/	Administrator-only commands
/home/	User directories
/etc/	System configuration files
/opt/	Specific tools installed by the sysadmin /usr/local/ often used instead
/root/	root user home directory
/mnt/	Mount points for temporarily mounted filesystems
/media	Mount points for removable media: /media/usbdisk, /media/cdrom

Linux filesystem structure (2)

Name	Description
/proc/	Access to system information /proc/cpuinfo, /proc/version ...
/sys/	System and device controls (cpu frequency, device power, etc.)
/lost+found	Corrupt files the system tried to recover
/tmp/	Temporary files
/usr/	Regular user tools (not essential to the system) /usr/bin/, /usr/lib/, /usr/sbin...
/usr/local/	Specific software installed by the sysadmin (often preferred to /opt/)
/var/	Data used by the system or system servers /var/log/, /var/spool/mail (incoming mail), /var/spool/lpd (print jobs)...

The Unix filesystem structure is defined by the Filesystem Hierarchy Standard (FHS): http://www.pathname.com/fhs/

Linux Basics

Shells and file handling

Command line interpreters

- **Shells:** tools to execute user commands
- Called “shells” because they hide the details on the underlying operating system under the shell's surface.
- Commands are input in a text terminal, either a window in a graphical environment or a text-only console.
- Results are also displayed on the terminal. No graphics are needed at all.
- Shells can be scripted: provide all the resources to write complex programs (variable, conditionals, iterations...)

Well known shells

Most famous and popular shells

- sh: The Bourne shell (obsolete)
  Traditional, basic shell found on Unix systems, by Steve Bourne.
- csh: The C shell (obsolete)
  Once popular shell with a C-like syntax
- tcsh: The TC shell (still very popular)
  A C shell compatible implementation with evolved features
  (command completion, history editing and more...)
- bash: The Bourne Again shell (most popular)
  An improved implementation of sh with lots of added features too.

Linux Command Syntax

command_name <arg1> [arg2]

- angle brackets for required parameters:
  - **Example:**
  - cp command syntax: cp [option] <Source> <Destination>
  - cp command Example: cp -r abc xyz
- square brackets for optional parameters:
  - **Example**:
  - ls command syntax: ls [option].. [filename] ..
  - ls command example: ls -a

ls command

Lists the files in the current directory, in alphanumeric order, except files starting with the "." character.

ls command Syntax: *ls [OPTION]... [FILE]...*

- ls -a (all)
  Lists all the files (including .* files)
- ls -l (long)
  Long listing (type, date, size, owner, permissions)
- ls -t (time)
  Lists the most recent files first
- ls -S (size)
  Lists the biggest files first
- ls -r (reverse)
  Reverses the sort order
- ls -ltr (options can be combined)
  Long listing, most recent files at the end

File name pattern substitutions

Better introduced by examples!

- ls *txt
  The shell first replaces *txt by all the file and directory names ending by txt (including .txt), except those starting with ., and then executes the ls command line.
- ls -d .*
  Lists all the files and directories starting with .
  -d tells ls not to display the contents of directories.
- cat ?.log
  Displays all the files which names start by 1 character and end by .log

The cp command

cp command syntax: (Copies the source file to the target)

cp [OPTION]... <SOURCE>... <DIRECTORY>

Case 1: Source & Destination both are Files.

1a. Destination file doesn’t exist. Create a destination file and copy the source file content.

Example: $ cp abc xyz (xyz file is empty file, abc file have some content)

1b. Destination file exist. Overwrite the source file content.

Example: $ cp abc 123 (abc & 123 file have some content).

The cp command

Case 2: Source is regular file & Destination is directory.

2a. Destination directory doesn’t exist. Show proper error.

2b. Destination directory exist and source files also already exist. Overwrite the existing files.

Example: (Assumption abc is a regular file located in test directory).

`$ cp abc test` (abc file overwrites)

`$ cp -i  abc test` (-i means interactive)
Asks for user confirmation if the target file already exists

`$ cp abc xyz test` (copy more than one file in to Directory)

Case3: Source is Directory and Destination should be a directory.

Example: `cp -r <source_dir> <target_dir>` (-r means recursive)
Copies the whole directory.

mv and rm commands

- mv command syntax: mv [option] <old_name> <new_name>

  Renames the given file or directory.
- mv -i (interactive)
  If the new file already exits, asks for user confirm
- rm file1 file2 file3 ... (remove)
  Removes the given files.
- rm -i (interactive)
  Always ask for user confirm.
- rm -r dir1 dir2 dir3 (recursive)
  Removes the given directories with all their contents.

Displaying file contents

Several ways of displaying the contents of files.

- cat file1 file2 file3 ... (concatenate)
  Concatenates and outputs the contents of the given files.
- more file1 file2 file3 ...
  After each page, asks the user to hit a key to continue.
  Can also jump to the first occurrence of a keyword
  (/ command).
- less file1 file2 file3 ...
  Does more than more with less.
  Doesn't read the whole file before starting.
  Supports backward movement in the file (? command).

The head and tail commands

- head [-<n>] <file>
  Displays the first <n> lines (or 10 by default) of the given file.
  Doesn't have to open the whole file to do this!
- tail [-<n>] <file>
  Displays the last <n> lines (or 10 by default) of the given file.
  No need to load the whole file in RAM! Very useful for huge files.
- tail -f <file> (follow)
  Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.
  Very useful to follow the changes in a log file, for example.
- Examples
  ```
  head windows_bugs.txt
  tail -f outlook_vulnerabilities.txt
  ```

The grep command

- grep <pattern> <files>
  Scans the given files and displays the lines which match the given pattern.
- grep error *.log
  Displays all the lines containing error in the *.log files
- grep -i error *.log
  Same, but case insensitive
- grep -ri error .
  Same, but recursively in all the files in . and its subdirectories
- grep -v info *.log
  Outputs all the lines in the files except those containing info.

Linux Basics

Command Documentation

Command help

Some Unix commands and most GNU / Linux commands offer at least one help argument:

- -h
  (- is mostly used to introduce 1-character options)
- --help
  (-- is always used to introduce the corresponding “long” option name, which makes scripts easier to understand)

You also often get a short summary of options when you input an invalid argument.

Manual pages

man <keyword>

Displays one or several manual pages for <keyword>

- man man

Most available manual pages are about Unix commands, but some are also about C functions, headers or data structures, or even about system configuration files!

- man stdio.h
- man fstab (for /etc/fstab)

Manual page files are looked for in the directories specified by the MANPATH environment variable.

Linux Basics

Users and Permissions

File access rights

Use ls -l to check file access rights

3 types of access rights

- Read access (r)
- Write access (w)
- Execute rights (x)

3 types of access levels

- User (u): for the owner of the file
- Group (g): each file also has a “group” attribute, corresponding to a given list of users
- Others (o): for all other users

Access right constraints

- **x** without **r** is legal but is useless
  You have to be able to read a file to execute it.
- Both **r** and **x** permissions needed for directories:
  x to enter, **r** to list its contents.
- You can't rename, remove, copy files in a directory if you don't have **w** access to this directory.
- If you have **w** access to a directory, you CAN remove a file even if you don't have write access to this file (remember that a directory is just a file describing a list of files). This even lets you modify (remove + recreate) a file even without w access to it.

Access rights examples

- -rw-r--r--
  Readable and writable for file owner, only readable for others
- -rw-r-----
  Readable and writable for file owner, only readable for users belonging to the file group.
- drwx------
  Directory only accessible by its owner

chmod: changing permissions

- chmod <permissions> <files>
  2 formats for permissions:
- Octal format (abc):
  a,b,c = r*4+w*2+x (r, w, x: booleans)
  Example: chmod 644 <file>
  (rw for u, r for g and o)
- Or symbolic format. Easy to understand by examples:
  chmod go+r: add read permissions to group and others.
  chmod u-w: remove write permissions from user.
  chmod a-x: (a: all) remove execute permission from all.

File ownership

Particularly useful in (embedded) system development when you create files for another system.

- chown -R sco /home/linux/src (-R: recursive)
  Makes user sco the new owner of all the files in /home/linux/src.
- chgrp -R empire /home/askywalker
  Makes empire the new group of everything in /home/askywalker.
- chown -R borg:aliens usss_entreprise/
  chown can be used to change the owner and group at the same time.

Beware of the dark side of root

- **root** user privileges are only needed for very specific tasks with security risks: mounting, creating device files, loading drivers, starting working, changing file ownership, package upgrades...
- Even if you have the root password, your regular account should be sufficient for 99.9 % of your tasks (unless you are a system administrator).
- In a training session, it is acceptable to use root.
  In real life, you may not even have access to this account, or put your systems and data at risk if you do.

In case you really want to use root...

- If you have the root password:
  su - (**s**witch **u**ser)
- In modern distributions, the sudo command gives you access to some root privileges with your own user password.
  Example: sudo mount /dev/hda4 /home

Linux Basics

Standard I/O, redirections, pipes

Standard output

More about command output

- All the commands outputting text on your terminal do it by writing to their *standard output*.
- **Redirection** means change the direction of input and output.
- Standard output can be written (redirected) to a file using the > symbol
- Standard output can be appended to an existing file using the >> symbol

Standard output redirection examples

- ls ~saddam/* > ~gwb/weapons_mass_destruction.txt
- cat obiwan_kenobi.txt > starwars_biographies.txt
  cat han_solo.txt >> starwars_biographies.txt
- echo “README: No such file or directory” > README
  Useful way of creating a file without a text editor.

Standard input

More about command input

- Lots of commands, when not given input arguments, can take their input from *standard input*.

-
  ```
  sort
  windows
  linux
  [Ctrl][D]
  linux
  windows
  ```
- `sort < participants.txt`
  The standard input of sort is taken from the given file.

Standard error

- Error messages are usually output (if the program is well written) to *standard error* instead of standard output.
- Standard error can be redirected through 2> or 2>>
- Example:
  `cat f1 f2 nofile > newfile 2> errfile`
- Note: 1 is the descriptor for standard output, so 1> is equivalent to >.
- Can redirect both standard output and standard error to the same file using &> :
  `cat f1 f2 nofile &> wholefile`

Pipes

Unix pipes are very useful to redirect the standard output of a command to the standard input of another one.

- Examples
  - cat *.log | grep -i error | sort
  - grep -ri error . | grep -v “ignored” | sort -u > serious_errors.log
  - cat /home/*/homework.txt | grep mark | more

Redirection vs Pipes

Redirection means change the direction of input and output.

Piping means change the flow of one command output to another command as input.

```
$ ls > log.txt
```

This command sends the output to the log.txt file

```
$ ls | grep file.txt
```

This command sends the output of the ls to grep command through the use of pipe(|), and the grep command searches for file.txt in the in the input provided to it by the previous command.

If you had to perform the same task using the first scenario, then it would be :-

```
$ ls > log.txt
$ grep 'file.txt' log.txt
```

So Pipe is used to send the output to other command whereas to redirect is used to redirect the output to some file.

Linux Basics

Looking for files

The find command

Better explained by a few examples!

- find . -name “*.pdf”
  Lists all the *.pdf files in the current (.) directory or subdirectories. You need the double quotes to prevent the shell from expanding the * character.
- find docs -name "*.pdf" -exec xpdf {} ';'
  Finds all the *.pdf files in the docs directory and displays one after the other.
- Many more possibilities available! However, the above 2 examples cover most needs.

The locate command

Much faster regular expression search alternative to find

- locate keys
  Lists all the files on your system with keys in their name.
- locate “*.pdf”
  Lists all the *.pdf files available on the whole machine
- locate “/home/fridge/*beer*”
  Lists all the *beer* files in the given directory (absolute path)
- locate is much faster because it indexes all files in a dedicated database, which is updated on a regular basis.
- find is better to search through recently created files.

Linux Basics

Compressing and archiving

Measuring disk usage

Caution: different from file size!

- du -h <file> (disk usage)
  -h: returns size on disk of the given file, in human readable format: K (kilobytes), M (megabytes) or G (gigabytes), .
  Without -h, du returns the raw number of disk blocks used by the file (hard to read).
  Note that the -h option only exists in GNU du.
- du -sh <dir>
  -s: returns the sum of disk usage of all the files in the given directory.

Measuring disk space

- df -h <dir>
  Returns disk usage and free space for the filesystem containing the given directory.
  Similarly, the -h option only exists in GNU df.

- Example:

```
> df -h .
Filesystem            Size  Used Avail Use% Mounted on
/dev/hda5             9.2G  7.1G  1.8G  81% /
```

- df -h
  Returns disk space information for all filesystems available in the system. When errors happen, useful to look for full filesystems.

Compressing and decompressing

Very useful for shrinking huge files and saving space

- g[un]zip <file>
  GNU zip compression utility. Creates .gz files.
  Ordinary performance (similar to Zip).
- b[un]zip2 <file>
  More recent and effective compression utility.
  Creates .bz2 files. Usually 20-25% better than gzip.
- [un]lzma <file>
  Much better compression ratio than bzip2 (up to 10 to 20%).
  Compatible command line options.

Archiving (1)

Useful to backup or release a set of files within 1 file

- tar: originally “tape archive”
- Creating an archive:
  `tar cvf <archive> <files or directories>`
  c: create
  v: verbose. Useful to follow archiving progress.
  f: file. Archive created in file (tape used otherwise).
- Example:

```
tar cvf /backup/home.tar /home
bzip2 /backup/home.tar
```

Archiving (2)

- Viewing the contents of an archive or integrity check:
  `tar tvf <archive>`
  t: test

- Extracting all the files from an archive:
  `tar xvf <archive>`

- Extracting just a few files from an archive:
  `tar xvf <archive> <files or directories>`
  Files or directories are given with paths relative to the archive root directory.

Linux Basics

Miscellaneous

Getting information about users

- who
  Lists all the users logged on the system.
- whoami
  Tells what user I am logged as.
- groups
  Tells which groups I belong to.
- groups <user>
  Tells which groups <user> belongs to.
- finger <user>
  Tells more details (real name, etc) about <user>
  Disabled in some systems (security reasons).

Misc commands (2)

- bc ("basic calculator?")
  bc is a handy but full-featured calculator. Even includes a programming language! Use the -l option to have floating point support.
- date
  Returns the current date. Useful in scripts to record when commands started or completed.

Modify time stamp of file

- touch command
  - Create empty file.
  - Modify timestamp of the file without change the content of the file.

Example:

```
$ ls -l abc
-rw-rw-r-- 1 kishore kishore 26 May 10 08:26 abc
$ date
Mon Jul  3 23:35:48 IST 2017
$ touch date
$ ls -l abc
-rw-rw-r-- 1 kishore kishore 26 Jul  3 23:35 abc
```

Changing users

You do not have to log out to log on another user account!

- su hyde
  (Rare) Change to the hyde account, but keeping the environment variable settings of the original user.
- su - jekyll
  (More frequent) Log on the jekyll account, with exactly the same settings as this new user.
- su -
  When no argument is given, it means the root user.

The wget command

Instead of downloading files from your browser, just copy and paste their URL and download them with wget!

wget main features

- http and ftp support
- Can resume interrupted downloads
- Can download entire sites or at least check for bad links
- Very useful in scripts or when no graphics are available (system administration, embedded systems)
- Proxy support (http_proxy and ftp_proxy env. variables)

wget examples

- wget -c http://139.59.40.228/KM_PGDESIoT_Brochure.pdf
  Continues an interrupted download.

- wget http://192.168.1.6/Linux_Basics.pdf

Misc commands (1)

- sleep 60

  Waits for 60 seconds
  (doesn't consume system resources).

- wc report.txt (word count)

  ```
  438  2115 18302 report.txt
  ```

  Counts the number of lines, words and characters in a file or in standard input.

Linux Basics

Symbolic Links

Symbolic links

A symbolic link is a special file which is just a reference to the name of another one (file or directory):

- Useful to reduce disk usage and complexity when 2 files have the same content.
- Example:
  anakin_skywalker_biography -> darth_vador_biography
- How to identify symbolic links:
  - ls -l displays -> and the linked file name.
  - GNU ls displays links with a different color.

Creating symbolic links

- To create a symbolic link (same order as in cp):
  ln -s file_name link_name
- To create a link with to a file in another directory, with the same name:
  ln -s ../README.txt
- To create multiple links at once in a given directory:
  ln -s file1 file2 file3 ... dir
- To remove a link:
  rm link_name
  Of course, this doesn't remove the linked file!

Hard links

- The default behavior for ln is to create *hard links*
- A *hard link* to a file is a regular file with exactly the same physical contents
- While they still save space, hard links can't be distinguished from the original files.
- If you remove the original file, there is no impact on the hard link contents.
- The contents are removed when there are no more files (hard links) to them.