Linux For Embedded Systems

For Arabs

Course 102:

Understanding Linux

Ahmed ElArabawy

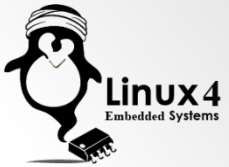
Lecture 12:
Basic Text Handling

Why Text Files

Text files play a big role in Linux such as,

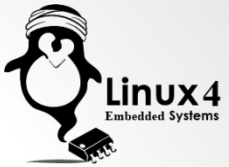
- All Linux configuration files are text based
- Source code for programs and scripts
- Program log files
- Web Pages (HTML files)
- XML files

In this Lecture,



- Displaying a text file
- Merging text files
- Creating a text file
- Emptying a text file
- Editing a text file
- Sorting text files
- Counting in text files
- Searching in text files
- Comparing text files
- Generating and Using Patches

Displaying a Text File (cat Command)



$ cat <file name>

This command sends the requested file to the standard output device (***stdout***) which is defaulted to be the screen

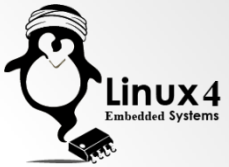
$ cat -n <filename>

This command is similar to the one above but also includes the line numbers in the output

$ cat <filename> | grep “string”

This command sends the file contents to the grep command which searches for the string, and sends the lines containing that string to ***stdout***

Displaying a Text File (more/less Commands)



- The cat command has a problem with long files, since the output is sent in bulk to the screen
- To fix that, we can use the commands more and less
- Both commands display the file in a page by page fashion

  $ less <filename>

  $ more <filename>

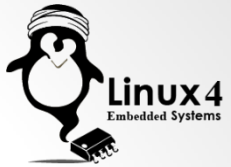
- The less command is better, since it allows the use of up/down scrolling)

- Very useful as the last command in a pipe,

  $ ls * | grep "log" | sort | less

  ls -la | less

Displaying a part of the Text File (head/tail Commands)



$ head <filename>
$ tail <filename>

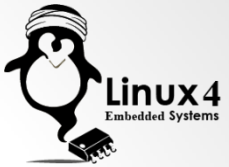
- Those commands list only the first (or the last) part of the file
  $ head file1 (displays first 10 lines of the file)
  $ tail file2 (displays the last 10 lines of the file)

- You can specify the number of lines to list
  $ head -100 file1 (displays the first 100 lines of the file)
  $ tail -50 file2 (displays the last 50 lines of the file)

- To list the updates of a file in a live fashion (very useful for tracking log files as they build up while the program is running)
  $ tail -f file1

- Very common log file to track
  $ tail -f -50 /var/log/messages

Concatenation of Files (cat Command ….. Again)



$ cat file1 file2 file3

This command concatenates the three files and send output to stdout

$ cat file1 file2 file3 > file.total

The three files are concatenated and stored in file.total

$ cat * > file.txt

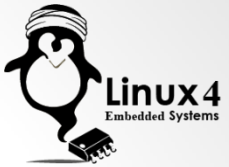
All files in the folder are merged in one file

Note,

If you have a folder containing short videos and want to concatenate them into a single movie, you can do,

$ cat *.mpeg > movie.mpeg

Creating an Empty Text File



- You can use redirection of an empty string to the file

  $ > file.txt

  Note that if file.txt exists, this will empty its contents (this is called truncating the file)

- You can use the ***touch*** command

  $ touch file.txt

  Note that if file.txt exists, this command will update the file time stamp without changing its contents

Creating/Appending Text Files

- We can create files in text editors, but we can quickly create ones using one of the following
  - Using echo,

    $ echo "Good Morning" > file1

    $echo "Good Morning" >> file2

    Although it looks weird, sometimes it is useful when creating a file inside a script

  - Using cat,

    $ cat > filename

    <put first line of text >

    …..

    <put last line of text>

    ^d

    Note that ^d stands for EOF (end of file)

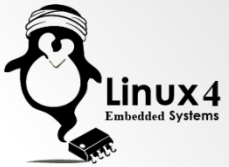
Editing a Text File

- GUI Based Editors:
  Most famous are:
  Emacs or ***xemacs*** (very powerful and complex)
  gedit (in Gnome)
  Kate (in KDE)
  gvim (Based on vi)

- Non GUI Based:
  Most famous is ***vi*** or ***vim*** (a must for any embedded developer)

- Note,
  - The editor vi (or vim) looks very primitive at the first use, but if you get used to it, you will find that it is an extremely powerful editor
  - Also, sometimes, vi is the only available editor on the embedded platform target. Hence, getting used to it is essential for embedded developers

Searching Text (grep Command)



$ grep <string or pattern> <files to search>

- It is a very powerful tool to search text
- It can search for a certain text pattern in one file, folder, or a tree of folders
- 'grep' can use the powerful regular expressions to improve search capability (to be discussed in a future lecture)

- Example

  $ grep "error" ./*.log

  $ grep "fatal error" .

  $ cat *.log | grep "error" | less

More on grep Command

$ grep -i <pattern> <files> (case insensitive search)

$ grep -r <pattern> <files> (recursive search)

$ grep -v <pattern> <files> (reverse search, show the lines not containing the pattern)

$ grep -n <pattern> <files> (show line numbers)

Examples:

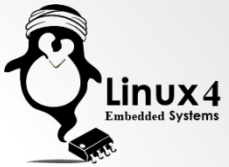
$ cat * | grep "error" | grep –v "fatal"

$ ls -l | grep "project"

$ grep -nr "printf" *.c

$ cat *.h | grep -n "#define"

Sorting Text (sort Command)



$ sort <file>

This command is used to sort lines in a file/files alphabetically

Examples:

$ sort file1 (sort the lines of the file and send it to stdout)

$ sort file1 > file2

$ sort -r file1 (reverse sort)

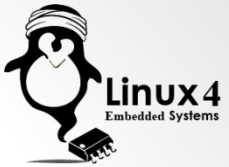
$ sort -u file1 (don't show similar lines)

Other Options:

- **-b** OR **--ignore-leading-blanks**
- **-f** OR **--ignore-case**
- **-n** OR **--numeric-sort** (99 is less than 100)

- Sort has other more advanced options, to do comparisons on specific fields for tabular data, for example to sort on file size of a long list of file details

Removing Repeated Lines (uniq Command)



$ uniq file1

This command is used to remove repeated lines in a file/files

This is a very common thing in log files

Examples:

$ uniq file1 (remove repeated lines and send output to stdout)

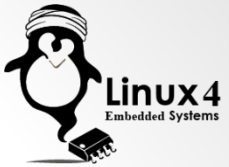
$ uniq -d file1 (only show duplicate lines)

Note:

- The command uniq only works on adjacent repeated lines, hence if you want to remove non-adjacent repeated lines, you must

  $ cat file1 | sort | uniq

Counting in Text Files (wc Command)



$ wc <file>

This command is used to count words/lines/chars in a file or files

Examples:

$ wc file1

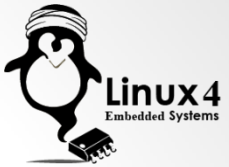
$ wc -l file1 (only line count)

$ wc -c file1 (only character count)

$ wc -w file1 (only word count)

$ cat * | wc -l

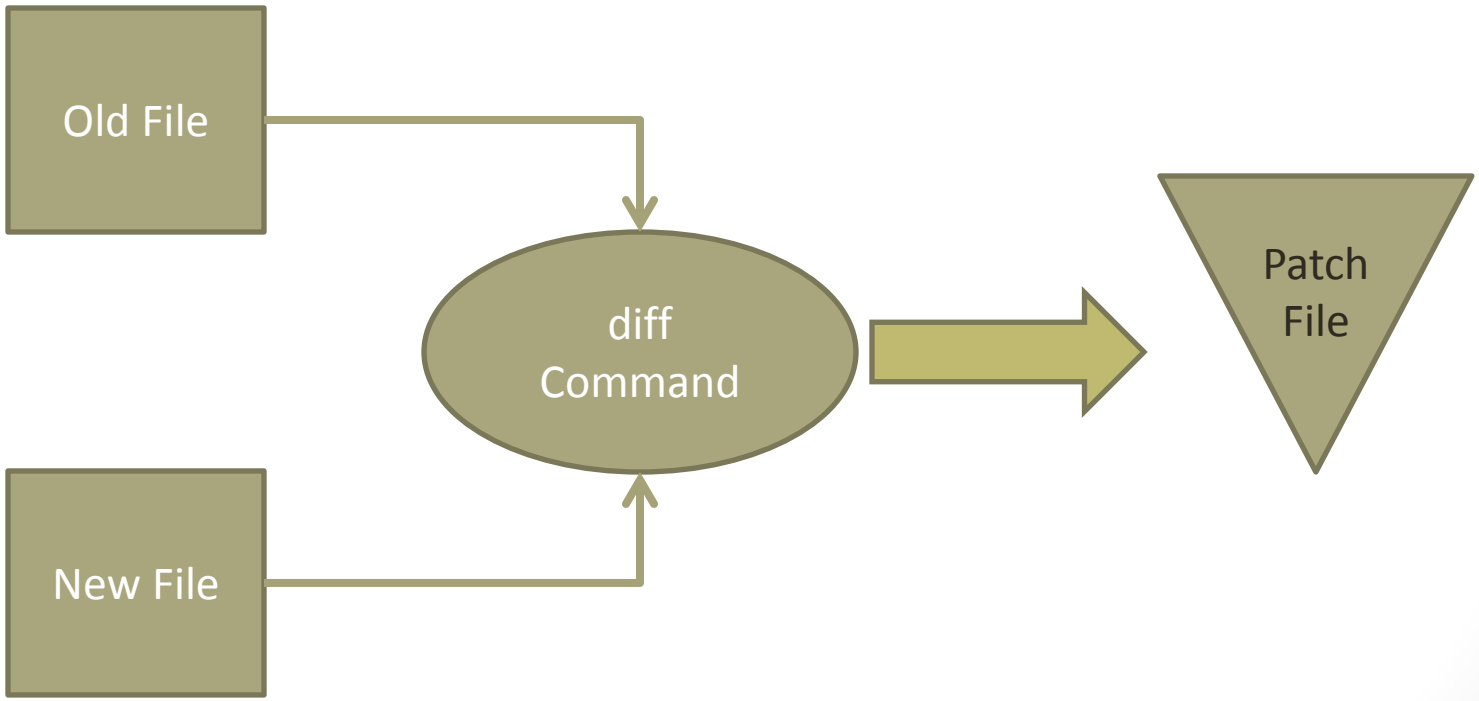
Comparing Text Files



- There are several GUI based tools for comparing text files in Linux, such as,
  - ***Meld***
  - ***Kdiff3***
- These tools are capable for both comparing two text files and merging changes in a third file
- This is very useful specially for use in source code files to identify changes between two versions of the same file
- We can also perform the comparison using the command line Interface using the command ***diff***

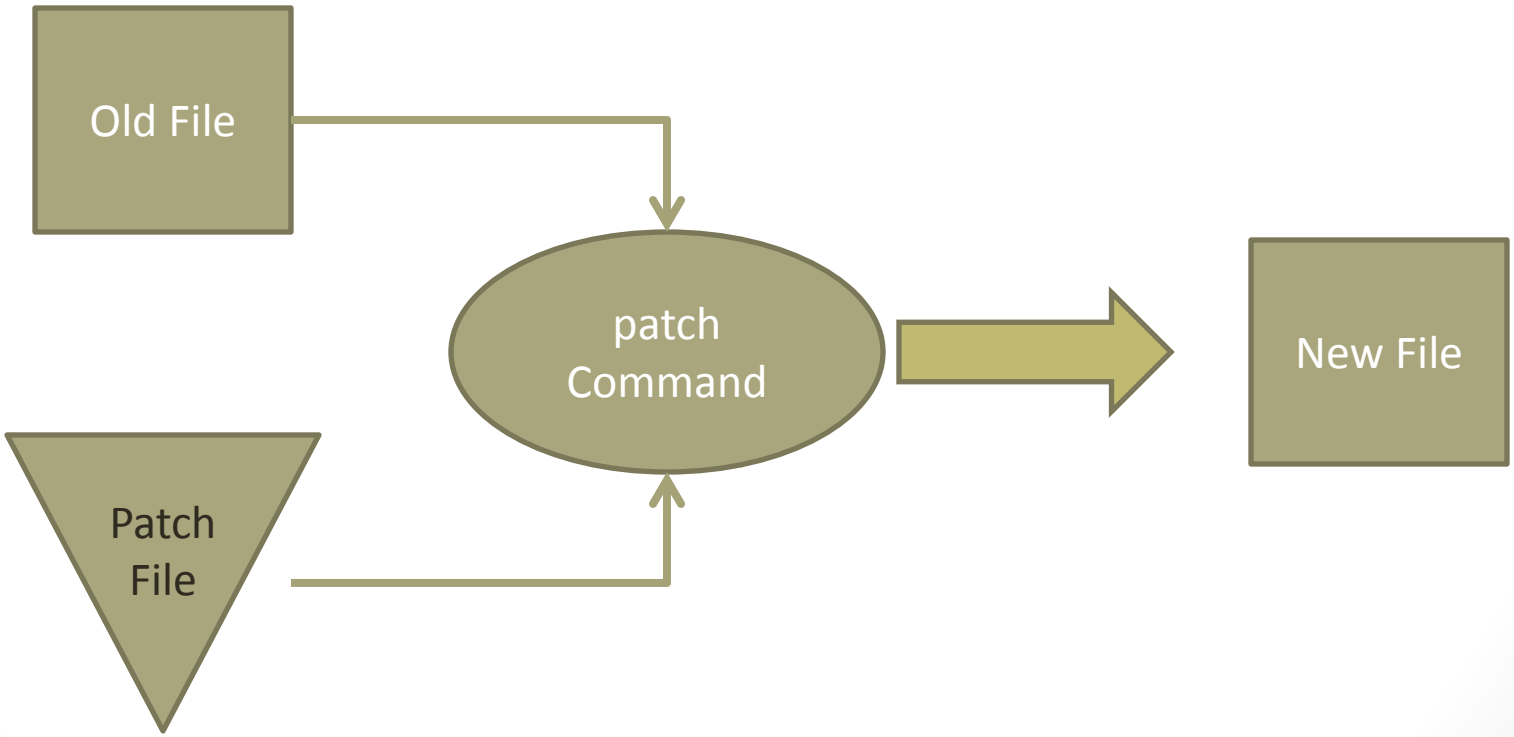
Comparing Text Files (diff Command)

- The ***diff*** command is very useful in comparison of files and for generation of patches (deltas between files) to be used later

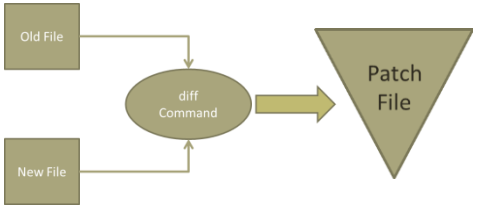


Patching Text Files (patch Command)

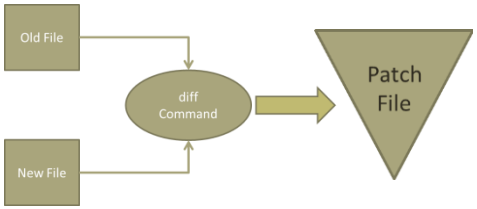
- The ***patch*** command is used to add the use the Patch file to update the old file to the new one

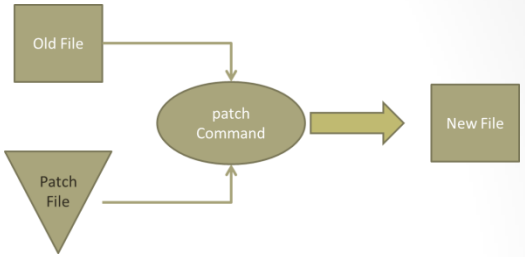


Generating and Using Patches

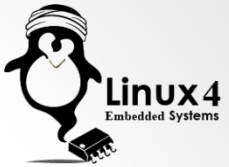


Generating and Using Patches





Comparing Text Files (diff Command)



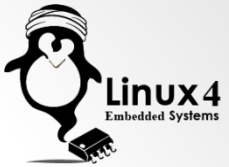
$ diff from-file to-file

This command compares two files as follows,

- If both **from-file** and **to-file** are filenames, then diff compares those two files
- If one of **from-file** & **to-file** is a filename, and the other is a directory, then diff compares the file with the one having the same name in the given directory
- If both **from-file** and **to-file** are directories, diff compares corresponding files in the two directories. The comparison can be recursive if "***-r***" is used
- Note that the output will go to ***stdout***, to send it to a patch file, you need to redirect the output

  $ diff old-file new-file > patch-file

Comparing Text Files (diff Command)



The ***diff*** command output can be one of the following,

- Normal Format

  $ diff <old-file> <new-file>

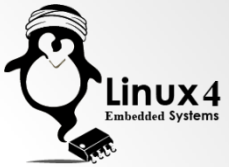
- Context Format

  $ diff -c <old-file> <new-file>

- Unified Format

  $ diff -u <old-file> <new-file>

‘diff’ Command
Normal Output Format



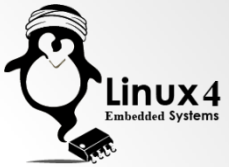
- The normal output consists of sections of text to be compared between the two files
- Each section is a line or more from the first file, compared to a line or more from the second file
- Then each section, there is a leading line (called the **Change Command**) to describe the main difference in this section. This line contains a letter that points out the difference
  - The letter ‘**a**’ stands for APPEND
  - The letter ‘**c**’ stands for CHANGE
  - The letter ‘**d**’ stands for DELETE
- So examples to the format for the Change Command is,

  8a12,15

  5,7c8,10

  5,7d3

‘diff’ Command Normal Output Format



8a12,15

Line 8 from the first file is appended by lines 12-15 from the second file

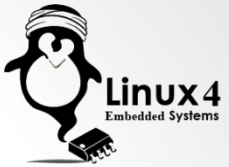
5,7c8,10

Lines 5-7 from the first file is replaced by lines 8-10 from the second file

5,7d3

Delete lines 5-7 from the first file (note, if they were not deleted they would have started at line 3 in the second file)

‘diff’ Command
Normal Output Format



After the Change Command, the patch file will contain corresponding lines from the first file, followed by a separator, then lines from the second file

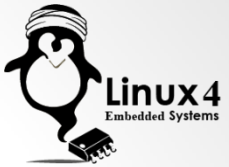
8a12,15
> This is the first line to be appended (line 12 in second file)
> This is the second line to be appended (line 13 in second file)
> This is the third line to be appended (line 14 in second file)
> This is the forth line to be appended (line 15 in second file)

5,6c8,10
< Line 5 in first file (will be deleted)
< Line 6 in first file (will be deleted)

> Line 8 of second file (will be added)
> Line 9 of second file (will be added)
> Line 10 of second file (will be added)

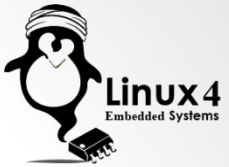
5,7d3
< Line 5 from first file (will be deleted)
< Line 6 from first file (will be deleted)
< Line 7 from first file (will be deleted)

‘diff’ Command Context Format



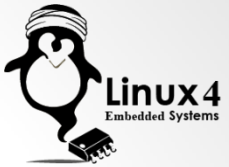
- In this format, the diff output more lines of each file to show the context. Some of these lines are not changed, but other have some changes
- The output starts with defining a symbol string for each file
- This is done by putting the symbol, followed by file name, followed by date of modification
- Then, there will be sections of text for file changes, in each section,
  - A line for each file to show line numbers to look at
  - A listing of the lines in each file
  - In front of each line, there is a code to show the state of that line
    - Nothing : means the line is the same in both files
    - ‘!’ : means the line has modification. This symbol will be in both files to show before and after state
    - ‘+’: means the line has been added
    - ‘-’ : means the line has been removed

‘diff’ Command Context Format



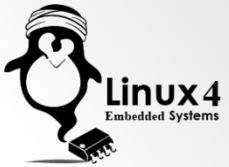

```
***     old-file        May     30      12:30:30  2014
---     new-file        June    1       07:12:40  2014
**********************
*** 3,7 ****
-  This is line #3 of first file (will be deleted in second file)
-  This is line #4 of first file (will be deleted in second file)
   This is line #5 (will be left unchanged)
!  This is line #6 (will be replaced with something else)
   This is line #7 (will be left unchanged)
--- 2,5 -----
   This is line#2 in second file (similar to line #5 in first file)
!  This is line#3 in second file (will replace line#6 in first file)
!  This is line#4 in second file (will replace line#6 in first file)
   This is line#5 in second file (similar to line #7 in first file)
```

‘diff’ Command Unified Format



- This output format tries to be more compact than the Context Format
- Instead of having 2 copies of listing (one per file), there will be one listing that shows lines in both files, and those which are removed from the first file, and those which are added to it
- The output starts by assigning a symbol (normally ‘**---**’) to the old file, and a symbol (normally ‘**+++**’) to the new file
- This is followed by a group of text sections in which there are changes between the files
- In each section,
  - Range of line numbers for each file
  - One listing of lines within that range with,
    - Starting with a ‘ ‘ means that the line is shared between the two files
    - Starting with a ‘**+**’ means that the line exists only in the second file (added)
    - Starting with a ‘**-**’ means that the line exists only in the first file (removed)

‘diff’ Command Unified Format



--- old-file May 30 12:30:30 2014

+++ new-file June 1 07:12:40 2014

@@ -3,5 +2,4 @@

- This is line #3 of first file (will be deleted in second file)

- This is line #4 of first file (will be deleted in second file)

This is line #5 of the first file(same as line#2 in second file)

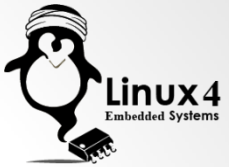
- This is line #6 (will be removed)

+ This is line#3 in second file (does not exist in first file)

+ This is line#4 in second file (does not exist in first file)

This is line #7 in first file (same as line#5 in second file)

Applying Patches
Patching A Single File



$ patch <original file> <patch file>

This command applies the patch to the original (old) file, to generate the new file

- By default, it tries to detect the format used in the patch file
- This automatic detection can be overridden by using

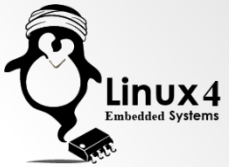
  $ patch -n <original file> <patch file>

  $ patch -c <original file> <patch file>

  $ patch -u <original file> <patch file>
- The patch is applied on the original file, and the file is updated
- To remove the patch (unpatch the file)

  $ patch -R <updated file> <patch file>

Applying Patches
Patching a Directory Tree



$ patch -p<num> < <patch-file>

- If the patch file was generated by comparing two directories

  $ diff -ru old-dir new-dir > patch-file

- Now we can use patch to update another directory with the old contents

  $ patch -p1 < patch-file

- We used ***-p1*** in this example assuming that the destination directory is on the same exact depth and name as the directory used in creating the patch directory and we are running the command from the
- If that is not the case, then we select the <num> based on how much of the directory name we want to strip
  - For example let us say the original directory inside the patch file has the path

    /home/tom/projects/alpha-project/src/….
  - And the destination directory is located at

    /home/bob/src/…
  - Then we do

    $ cd /home/bob

    $ patch -p5 < patch-file

Linux 4 Embedded Systems

http://Linux4EmbeddedSystems.com