

File Manipulations

File Manipulations

- The information is organized in files and directories. A directory is a collection of files and other directories. Information about these files and directories are stored in a tree of **inodes**. The inode contains information about where to access the file, the access permissions for the file, and an inode number.
- The inodes are structured as a hierarchical tree with a single root directory (“/”) at the top.

- Multiple disk drives and other devices can be 'mounted' at a directory along with the tree (E.g. the first disk drive may be mounted to the root directory – “/”). These devices will have their own physical file system for organizing the contained information (E.g. ext4 – the fourth extended filesystem is a filesystem that is used in Linux).
- It is also possible to mount virtual filesystems along the tree (E.g. the /proc directory is used to access the information about processes).

Unix supports a number of file types

#1) Ordinary Files

These files contain binary or text information and are stored in a directory on a disk drive.

#2) Directory Files

These are used to organize a group of files – the contained files may be of any type.

#3) Special Files

Special files, also known as device files, are used to represent physical devices such as a printer, a disk drive, or a remote terminal.

#4) Named Pipes

Named pipes are used to allow one process to send information to another. These are temporary files that hold information from one process until it is read by another process.

#5) Symbolic Links

These are the files that reference some other file or directory with an absolute or relative path.

The 'ls' command is used to list filenames and other associated data. With the option 'ls -il', this command lists out a long format of file details along with its inode number.

#1) **chmod**: Change file access permissions.

- **Description:** This command is used to change the file permissions. These permissions read, write and execute permission for owner, group, and others.
- **Syntax (symbolic mode):** *chmod [ugoa][[+ -=][mode]] file*
- The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll.
- The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) a permission.
- The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.
- **Example:** Add write permission for user, group and others for file1.
- *\$ chmod ugo+w file1*

chmod

- **Syntax (numeric mode):** *chmod [mode] file*
- The mode is a combination of three digits – the first digit indicates the permission for the user, the second digit for the group, and the third digit for others.
- Each digit is computed by adding the associated permissions. Read permission is '4', write permission is '2' and execute permission is '1'.
- **Example:** Give read/write/execute permission to the user, read/execute permission to the group, and execute permission to others.
- *\$ chmod 751 file1*

#2) **chown**: Change ownership of the file.

- **Description**: Only the owner of the file has the rights to change the file ownership.
- **Syntax**: `chown [owner] [file]`
- **Example**: Change the owner of file1 to user2 assuming it is currently owned by the current user
 - *\$ chown user2 file1*

#3) **chgrp**: Change the group ownership of the file

- **Description:** Only the owner of the file has the rights to change the file ownership
- **Syntax:** `chgrp [group] [file]`
- **Example:** Change group of file1 to group2 assuming it is currently owned by the current user
 - *\$ chgrp group2 file1*

Different ways of comparing two files in Unix

- **#1) cmp:** This command is used to compare two files character by character.
- **Syntax:** `cmp [options] file1 file2`
- **Example:** Add write permission for user, group and others for file1.
 - *\$ cmp file1 file2*

- **#2) comm:** This command is used to compare two sorted files.
- **Syntax:** *comm [options] file1 file2*
- One set of options allows selection of 'columns' to suppress.
 - -1: suppress lines unique to file1 (column 1)
 - -2: suppress lines unique to file2 (column 2)
 - -3: suppress lines common to file1 and file2 (column3)
- **Example:** Only show column-3 that contains lines common between file1 and file2
 - *\$ comm -12 file1 file2*

- **#3) diff:** This command is used to compare two files line by line.
- **Description:** The output indicates how the lines in each file are different, and the steps involved to change file1 to file2. The 'patch' command can be used to make the suggested changes. The output is formatted as blocks of:
 - **Change commands**
 - *< lines from file1*
—
> lines from file2

- The change commands are in the format [range][acd][range]. The range on the left may be a line number or a comma-separated range of line numbers referring to file1, and the range on the right similarly refers to file2. The character in the middle indicates the action i.e. add, change or delete.
- 'LaR' – Add lines in range 'R' from file2 after line 'L' in file1.
- 'FcT' – Change lines in range 'F' of file1 to lines in range 'T' of file2.
- 'RdL' – Delete lines in range 'R' from file1 that would have appeared at line 'L' in file2

- **Syntax:** *diff [options] file1 file2*
- **Example:** Add write permission for user, group and others for file1
 - *\$ diff file1 file2*

- **#4) dircmp:** This command is used to compare the contents of directories.
- **Description:** This command works on older versions of Unix. In order to compare the directories in the newer versions of Unix, we can use `diff -r`
- **Syntax:** *dircmp [options] dir1 dir2*
- **Example:** Compare contents of dir1 and dir2
 - *\$ dircmp dir1 dir2*

- **#5) uniq:** This command is used to filter the repeated lines in a file which are adjacent to each other
- **Syntax:** *uniq [options] [input [output]]*
- **Example:** Omit repeated lines which are adjacent to each other in file1 and print the repeated lines only once
 - *\$ uniq file1*

Unix Filename Wildcards – Metacharacters

- **#1) ‘*’ – any number of characters:**
- This wild-card selects all the files that matches the expression by replacing the asterisk-mark with any set of zero or more characters.
- **Example1:** List all files that start with the name ‘file’. g. file, file1, file2, filenew
– \$ ls file*
- **Example2:** List all files that end with the name ‘file’. g. file, afile, bfile, newfile
– \$ ls *file

- **#2) '?' – single character:**
- This wild-card selects all the files that matches the expression by replacing the question-mark with any one character.
- **Example1:** List all files that have one character after 'file'. g. file1, file2, filea
 - \$ ls file?
- **Example2:** List all files that have two characters before 'file'. g. dofile, tofile, a1file
 - \$ ls ??file

- **#3) '[' *range* ']' – single character from a range:**
- This wild-card selects all the files that matches the expression by replacing the marked range with any one character in the range.
- **Example1:** List all files that have a single digit after 'file'. g. file1, file2
 - \$ ls file[0-9]
- **Example2:** List all files that have anyone letter before 'file'. g. afile, zfile
 - \$ ls [a-z]file

- # 4) '[' *range* ']*' – multiple characters from a range:
- This wild-card selects all the files that matches the expression by replacing the marked range with one or more characters from the range.
- **Example1:** List all files that have digits after 'file'. g. file1, file2, file33
– \$ ls file[0-9]*

Unix Regular Expressions

- Regular expressions can be used with text processing **commands like *vi, grep, sed, awk, and others***. Note that although some regular-expression patterns look similar to filename-matching patterns – the two are unrelated.
- **#1) ‘^’ – anchor character for start of line:**
- If the carat is the first character in an expression, it anchors the remainder of the expression to the start of the line.
- **Example1:** Match all lines that start with ‘A’. g. “A plane”
 - Pattern: ‘^A’
- **Example2:** Match all lines that start with ‘hello’. g. “hello there”
 - \$ grep “^hello” file1

- **#2) '\$' – anchor character for end of line:**
- If the carat is the last character in an expression, it anchors the remainder of the expression to the end of the line.
- **Example1:** Match all lines that end with 'Z'. g.
"The BUZZ"
 - Pattern: 'Z\$'
- **Example2:** Match all lines that end with 'done'. g.
"well done"
 - \$ grep "done\$" file1

- **#3) ‘** – any single character:
- The ‘ character matches any character except the end-of-line.
- **Example1**: Match all lines that contain a single character. g. “a”
 - **Pattern**: ‘^.\$’
- **Example2**: Match all lines that end with ‘done’. g. “well done”
 - \$ grep “done\$” file1

- **4) '[' range ']' – a range of characters:**
- This pattern matches the set of characters specified between the square brackets.
- **Example1:** Match all lines that contain a single digit. g.
"8"
 - **Pattern:** '^[0-9]\$'
- **Example2:** Match all lines that contain any of the letters 'a', 'b', 'c', 'd' or 'e'
 - \$ grep "[abcde]"
- **Example3:** Match all lines that contain any of the letters 'a', 'b', 'c', 'd' or 'e'.
 - \$ grep "[a-e]" file1

- **#5) ‘[^’ *range* ’]** – a range of characters to be excluded:
- This pattern matches any pattern except the set of characters specified between the square brackets.
- **Example1**: Match all lines that do not contain a digit. g. “hello”
 - **Pattern**: ‘[^0-9]’
- **Example2**: Match all lines that do not contain a vowel
 - `$ grep “[^aeiou]” file1`

- **#6) '*' – 'zero or more' modifier:**
- This modifier matches with zero or more instances of the preceding character-set.
- **Example1:** Match all lines that contain 'ha' followed by zero or more instances of 'p' and then followed by 'y'. g. "happpy" or "hay"
 - **Pattern:** 'hap*y'
- **Example2:** Match all lines that start with a digit following zero or more spaces E.g. " " or "2."
 - \$ grep "[0-9]" file1

- **#7) ‘?’ – ‘zero or one’ modifier:**
- This modifier matches with zero or one instances of the preceding character-set.
- **Example1:** Match all lines that contain ‘hap’ followed by zero or one instances of ‘p’ and then followed by ‘y’. g. “hapy” or “happy”
 - **Pattern:** ‘happ?y’
- **Example2:** Match all lines that start with a digit followed by zero or one ‘:’ characters E.g. “1” or “2:”
 - `$ grep “[0-9]:?” file1`

Thank You