Commands

Sunday, 4 February 2024 18:01

1. **/** (Forward Slash):
   - The forward slash `/` is used as a directory separator in file paths. It separates directory names in a hierarchical file system.
   - For example, in the path `/home/user/file.txt`, `/` separates the root directory from the `home` directory, and so on.
2. **~** (Tilde):
   - The tilde `~` is a shorthand notation representing the home directory of the current user.
   - For example, if the username is `user`, then `~` refers to the directory `/home/user`.
   - It's often used in commands and scripts to refer to files or directories within the user's home directory without explicitly typing out the full path.

For the root user, home is at /root
Regular users, it is /home/username (e.g. /home/kc)

Pwd :present working directory.
Cd : change directory.

. Current directory
.. Previous directory

ls: display contents of the current directory .

Syntax: ls [OPTIONS] [FILES]

ls -l long listing format .

ls -a display hidden files also.

ls -x used to find type of files(like .txt , .jpg etc

ls -x *.txt gives all files in the given directory of the type .txt

ls -s gives size of each file in the given directory

ls -S gives files in descending order of their size

ls -t shows recently opened file in the top

ls -lt shows recently opened file in the bottom

ls -R lists the contents of a directory and all of its subdirectories, recursively.

mkdir (make directory):

Syntax: mkdir [OPTION] [DIRECTORY]

mkdir filename creates folder filename in the pwd

mkdir -p a/b creates directory b and a, b is in directory a

mkdir a/b created directory b in a provided directory a exists already

echo used to display the text passed in as an argument

echo "Hello, world!" prints in terminal

echo "Hello, world!" > message.txt creates message.txt and "Hello, world!" is printed in it

Cat (concatenation): disp

lay the contents of a text

file to screen

```
-n      Number the output lines, starting at 1.

-s      Squeeze multiple adjacent empty lines, causing the output to be
        single spaced.
```

mv: move files or folders

mv <source> <destination> ex: mv myfile.txt newfile.txt

mv -i

mv -i myfile.txt newfile.txt if newfile.txt exists it asks us to overwrite or not

if newfile.txt doesn't exists it renames myfile.t
xt to newfile.txt

mv -n

mv -n file1 file2 if file2 exists it gives warning

if it doesn't it renames file1 to file 2

mv -f forcibly overwrites files , doesn't ask for conformation.

SOURCE can be one, or more files or directories, and DESTINATION can be a single file or directory
When multiple files or directories are given as a SOURCE, the DESTINATION must be a directory

- In this case, the SOURCE files are moved to the target directory.

If you specify a single file as SOURCE, and the DESTINATION target is an existing directory, then the file is moved to the specified directory.
If you specify a single file as SOURCE, and a single file as DESTINATION target then you're renaming the file .
When the SOURCE is a directory and DESTINATION doesn't exist, SOURCE will be renamed to DESTINATION. Otherwise if DESTINATION exist, it be moved inside the DESTINATION directory.

cp similar to mv excepts it copies instead of moving
cp -R copies recursively everything in the directory
cp -p preserves everything like time , user id etc.

rm removes permanently i.e it doesn't send it to trash.
rm -i asks to remove or not.
rmdir / rm -d removes folder provided folder is empty.
rm -r recursively removes everything in that directory.

```
head [-n count | -c bytes] [file ...]

RIPTION
This filter displays the first count lines or bytes of each of the
specified files, or of the standard input if no files are specified.  If
count is omitted it defaults to 10.

The following options are available:

-c bytes, --bytes=bytes
        Print bytes of each of the specified files.

-n count, --lines=count
        Print count lines of each of the specified files.
```

Regular Expressions (regex)

- Used in text editors, programming languages, and command-line tools (grep, sed, awk etc)
- regex: a pattern that matches a set of strings
- Metacharacters: characters with special meaning
  - "^" beginning of a line (Can also mean "not" if inside [])
  - "$" end of line
  - "." match any single character
  - "\" escape a special character
  - "|" or operation i.e. match a particular character set

on either side

metacharacters

Metacharacters are special characters that have a special meaning to the shell. They are used to perform various operations on files and directories. Some of the most commonly used metacharacters are:

***:**

The asterisk metacharacter can be used anywhere in the filename. It does not necessarily have to be the last character. The * metacharacter can be used to match any number of characters. For example, the command ls *.txt will list all files in the current directory that have the .txt extension.

?:

The question mark metacharacter is used to match a single character in a filename. For example, the command ls a?b.txt will list all files in the current directory that have the .txt extension and start with the letter a and followed by a single character and then the letter b. doesnt show a.txt

[]:

The square bracket metacharacters can be used to match a set of characters. For example, the command ls [a-z].txt will list all files in the current directory that have the .txt extension and start with a lowercase letter.
doesnt show ab.txt

^:

The caret metacharacter can be used to match the beginning of a line. For example, the command grep "^hello" file.txt will print all lines in the file file.txt that start with the word "hello".

$:

The dollar sign metacharacter can be used to match the end of a line. For example, the command grep "hello$" file.txt will print all lines in the file file.txt that end with the word "hello".

Quantifiers: specifying the number of occurrences of a character

- "*" Match the preceding item zero or more times
- "?" Match the preceding item zero or one time
- "+" Match the preceding item one or more times
- "{n}" Match the preceding item exactly n times
- "{n,} Match the preceding item at least n times
- "{,m}"Match the preceding item at most m times
- "{n,m}" Match the preceding item from n to m times

[^abc] matches any character except 'a', 'b', or 'c'.
[^0-9] matches any non-digit character.
[^a-zA-Z] matches any non-alphabetic character.

grep

grep: “global regular expression print”

- Searches files for a particular pattern of characters

Grep searches pattern in the files but not files itself.
If you cannot see colors use --color
grep -I ignores the case(case in sensitive)
grep -i "for" bigfile
Metacharacters can be used

```
-E, --extended-regexp
        Interpret pattern as an extended regular
        expression (i.e., force grep to behave as
        egrep).
```

(searches for "for" in the bigfile and prints it)
grep "cried|could" bigfile
grep -E "cried|could" bigfile
grep "cried\|could" bigfile

1. Gives error

2,3 prints with "cried" or "could".

\s - whitespace
\b matches any character that is not a letter or number without including itself in the match.

grep -v negation
-r recursive search, -w whole word, -n show line number, -c count (count of lines, not number of occurrences).

find

Locates files/directories

Find <directory> <file type> <pattern>
-name tells to match the name of the given file
-iname case insensitive
-type f check only for files (-type f)
 find . -type d lists all directories in the current directory
-size +60k files of size more than 60kbytes
find . -perm 644 shows files with permission set to 644

wc

Default output: lines, words, byte-count
wc prints word (-w), character (-m), line (-l), byte-count (-c) in a file

cut

Cut -d',' -f 1,3 <filename>
The first command specifies what delimiter to use (, in this case) and to output fields 1 and 3 (serial no and name)
 --output-delimiter=" "   change the output delimiter
Cut -b. With respect to bites

paste

Helps merge lines of files horizontally
-d can be used to specify a delimiter when pasting
Paste -d'_' <file><file>
-s pastes one file at a time in serial.

sort

Sort ascending order (13 will come before 9)
Sort -r reverse order
sort -n. numerically sort
sort -t ',' -k8 <file>
  -t ',' delimiter ,
   -k8 8 th field in the file separated by ,

zip

Don't forget -r recursive zip
Zip -r <destiantion name> <files>
-e asks password

unzip

Unzip -l allows to view without unzipping
unzip example.zip -d dir1/ unzips in dir1

Ps and kill

ps displays a list of processes currently executing ps in a shell without any options
 aux, "a": display the processes of all users; "u": user-oriented formats; "x": list processes without a controlling terminal

Access control

r (read) = 4
w (write) = 2
 x (execute) = 1
Chmod to change permissions