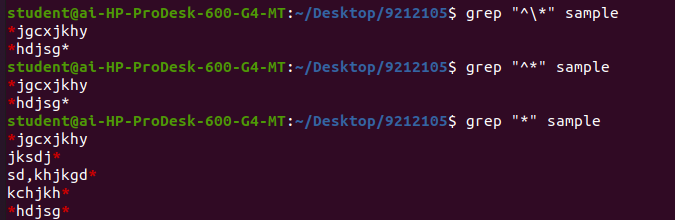
Exercise -1

10.2



10.3

grep -ri --include=\*.html -o "IMG SRC" . | wc -l

grep -ri performs a case-insensitive (-i) recursive (-r) search for the

given pattern in the current directory (.)

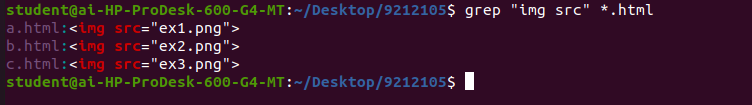
--include=\*.html limits the search to only HTML files

-o option only prints the matched string, rather than the entire line

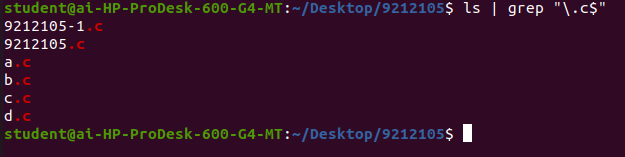
that contains it

wc -l counts the number of lines output by grep, giving the total count

of lines containing the string "IMG SRC".



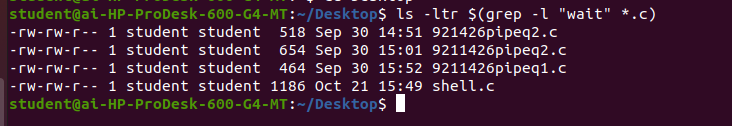
10.4



10.5



10.6



10.7



(i)a.\*b=The regular expression a.\*b matches any string that starts with an "a"

followed by zero or more characters (.) and ends with a "b".

(ii)..\*=it matches any string that has two or more characters.

(iii)^{$=The regular expression ^{$ matches an empty string.

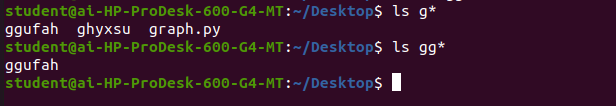
10.8



10.9

g\*=matches zero or more occurrences of the letter "g".

gg\*=matches zero or more occurrences of “gg”.



10.10

^.{10,}$

^.\*\{10,\}$

10.12

This command uses grep and cut to extract the name of the users who have

their default shell specified in the /etc/passwd file.

* grep "$SHELL$" /etc/passwd: This command uses the grep utility to

search the /etc/passwd file for the line that exactly matches the

current shell of the system, specified by the environment variable

$SHELL. The $ characters are escaped with a backslash to prevent shell

expansion, so the grep command searches for the literal string

"$SHELL$".

* cut -d: -f1: This command uses the cut utility to extract the first field

(field 1) from the input, which is delimited by colons (:). The first field in

each line of the /etc/passwd file is the username.

So, this command will return a list of usernames for users who have their

default shell set to the current shell of the system.

10.13

grep -h -E '^[0-9]+\.[0-9]+(\.[0-9]+)?' chap\* | sort -n | uniq

10.14





10.15



10.16





10.17





10.18

sed 's/linux/red hat linux/g' filename

This is the command to replace linux with red hat linux.



10.19



Exercise -2

10.1

A wild card and a regular expression are both used to match patterns in text,

but they differ in the level of complexity and flexibility they offer.

Wild cards are typically used for simple pattern matching, such as when

searching for files with a certain name or extension.

With regular expressions, you can specify patterns that match specific

character sets, repeat characters, match the beginning or end of a line, and

much more.

10.2

(i) grep a b c: This command searches for the string "a" in the files "b" and "c".

If "b" and/or "c" contain the letter "a", then grep will output the line

containing the string "a".

(ii) grep <HTML> foo: This command searches for the string "<HTML>" in the

file "foo". If "foo" contains the string "<HTML>", then grep will output the line

containing that string.

(iii) grep "\*\*" foo: This command searches for the string "" in the file "foo". If

"foo" contains the string "", then grep will output the line containing that

string.

(iv) grep \*: This command is not complete as it is missing an argument, so it

will result in a syntax error. However, if a valid argument is provided, this

command will search for any string that matches the pattern specified by the

asterisk (\*).

10.3

the \*.htm\* pattern matches all files in the current directory that have a file

name extension of ".htm" or ".html" . This means that the command will

search for the string 'botswana.\*birds' in all files in the current directory that

have an HTML file name extension.

The regular expression 'botswana.\*birds' is then used to match any line that

contains the word "botswana" followed by zero or more characters, and then

the word "birds". The .\* in the expression is a regular expression that matches

any character zero or more times.

So, the \* in the command plays a key role in allowing the command to search

for the specified pattern in multiple files at once, rather than having to

search through each file individually.

10.4

The command grep '.\*' foo will display all the lines in the file foo since the

regular expression '.\*' matches any sequence of characters (including an

empty sequence) in a line.

If the quotes are removed from the command and it is run as grep .\\* foo,

the shell may interpret the \* character as a special character and expand it to

the list of all file names in the current directory. This would cause grep to try

to search through all the files in the current directory, which is likely not

what was intended. Therefore, it is generally a good practice to enclose

regular expressions in quotes or escape special characters when using them

in the shell.

10.5

The command grep -l "echo '\t'" foo is used to search for lines containing a

tab character in the file foo, and print out only the names of the files that

contain at least one matching line.

10.6

No, the two commands are not equivalent, they have opposite meanings.

The command grep "^[^a-z]" foo searches for lines in the file "foo" that start

with a character that is not a lowercase letter, and matches any line that has

a character other than a lowercase letter at the beginning of the line. The

regular expression ^[^a-z] means "match any line that starts (^) with any

character that is not ([^...]) a lowercase letter (a-z)".

On the other hand, the command grep -v "^[a-z]" foo searches for lines in

the file "foo" that do not start with a lowercase letter. The -v option inverts

the search, meaning it returns all lines that do not match the regular

expression. The regular expression ^[a-z] means "match any line that starts

(^) with a lowercase letter (a-z)".

10.7

grep -vE "^\s\*(#|/\\*|//)" filename

This command is used to identify all nonblank lines in a file.

10.8

grep "$(date +%m/%d/%Y)" emp.lst | awk '{print $1 " " $2}'

$(date +%m/%d/%Y) runs the date command and uses the +%m/%d/%Y

format specifier to generate today's date in the format mm/dd/yyyy. This

date string is then used as the pattern for grep.

grep "$(date +%m/%d/%Y)" emp.lst searches for all lines in the emp.lst file

that contain today's date, using the date pattern generated by date.

awk '{print $1 " " $2}' processes the output from grep and prints the first and

second fields (separated by spaces) from each matching line, which should be

the names of the persons born today.

10.10

grep fork \*.c searches for lines containing the word "fork" in all .c files in the

current directory and outputs them to stdout.

cut -d: -f1 extracts the first field from each line of the grep output,using the

colon as the delimiter.

sort -u sorts and removes duplicates from the list of filenames.

ls -t lists the files in the current directory in order of modification time, with

the most recently modified file listed first.

So, the entire pipeline lists the .c files in the current directory that contain

the word "fork", sorted by modification time with the most recently modified

file listed first.

ls -t $(grep -l fork \*.c)

This is the code which uses two commands.

grep -l fork \*.c searches for all .c files in the current directory that contain

the word "fork", and outputs only the names of the matching files to stdout,

one per line.

ls -t lists the files in the current directory in order of modification time, with

the most recently modified file listed first. The list of files is provided as

arguments to ls using command substitution ($(...)), which runs the grep

command first and uses its output as the arguments to ls.

Exercise 3

10.12

(i)` jeff(er(y|ies|ey)s)`

(ii)` hitch(e|i)(n|ng)`

(iii) `[Hh](eard|erd|ird)`

(iv)` dix(on|ons?|k|ks?)`

(v)` Mc?Gee`

(vi) `wood(cock|house)?`

10.13

(i)[0-9]\* matches zero or more occurrences of any digit from 0 to 9. This

means it can match an empty string, or any string containing only digits.

[0-9][0-9]\* matches one or more occurrences of any digit from 0 to 9. This

means it cannot match an empty string, but can match any string containing

only digits or any string that starts with one or more digits.

(ii)^[^^] matches any string that starts with a character that is not a caret

(^), followed by any other character. This means it will not match any string

that starts with a caret.

^^^ matches exactly three carets (^). This means it will only match the

string "^^^"

10.15

find . -maxdepth 1 -type f ! -writable -ls

. specifies the current directory as the starting point for the search.

-maxdepth 1 limits the search to only the current directory, and doesn't

descend into subdirectories

-type f restricts the search to only files, and not directories or other types of

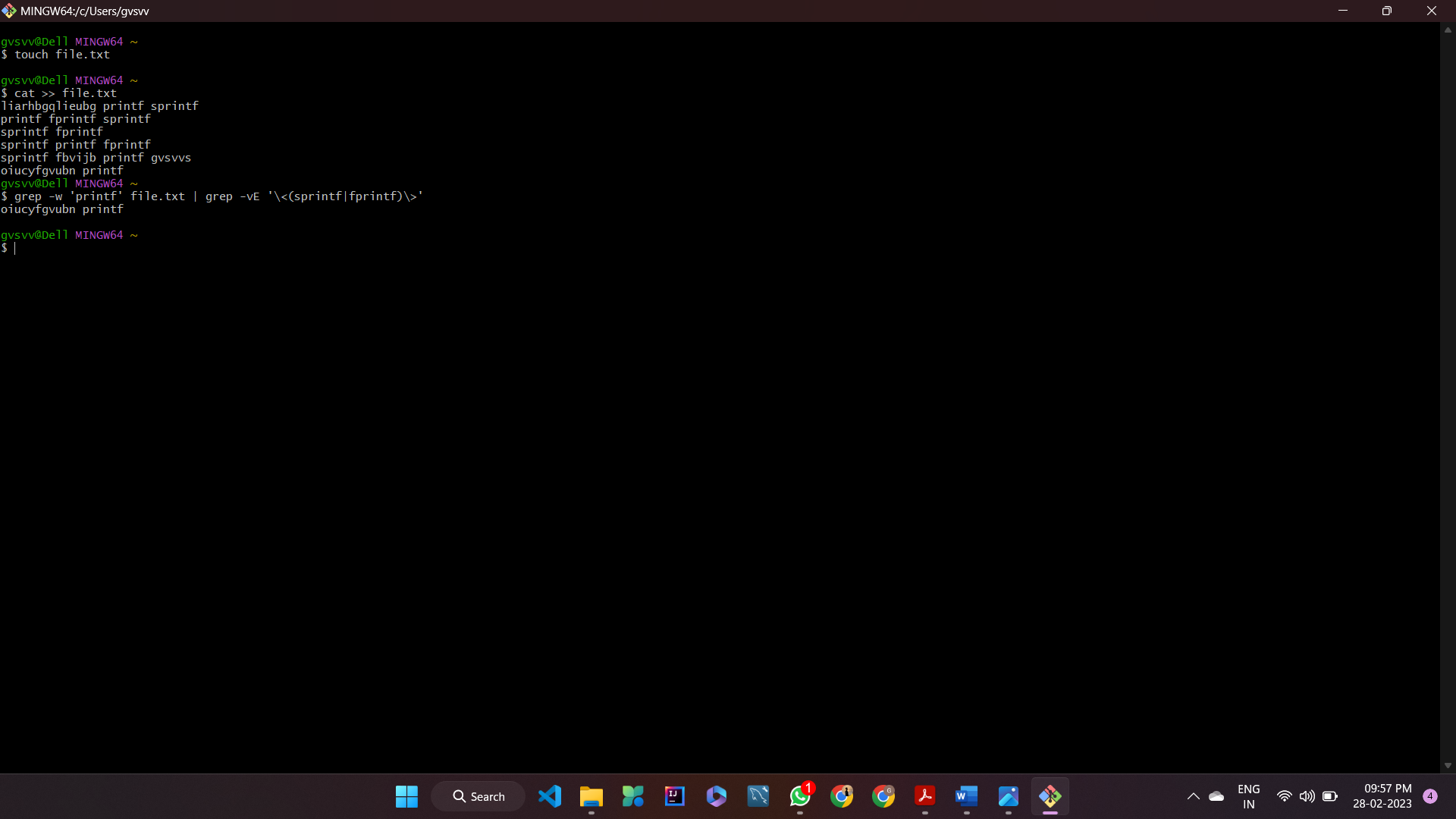
files.

! -writable specifies that the files listed should not be user-writable.

-ls prints a detailed listing of each file, including its permissions and other

metadata

10.16



10.18

grep '\b(Bill|William) (Christie|Christy)\b' filename

10.19

find $HOME -name '\*.c' -exec grep -l

'int[[:space:]]\+main\s\*(\|int[[:space:]]\+main(' {} \; | xargs -n1 vi

find $HOME -name '\*.c': Searches for all files ending in .c in the home

directory and its subdirectories.

-exec grep -l 'int[[:space:]]\+main\s\*(\|int[[:space:]]\+main(' {} \;: Executes

the grep command on each found file, searching for the regular expression

int[[:space:]]\+main\s\*(\|int[[:space:]]\+main( which matches all

occurrences of int main( or int main( or int main (.

| xargs -n1 vi: Passes the list of found files to the xargs command, which then

opens each file in turn in the vi editor.

10.20

sed -i '1i<HTML>' -e '$a<\/HTML>' file.html

-i: Specifies that the changes should be made to the file in place, instead of

printing them to standard output.

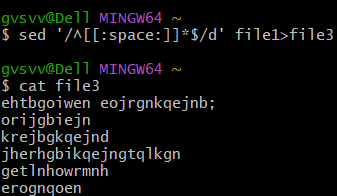
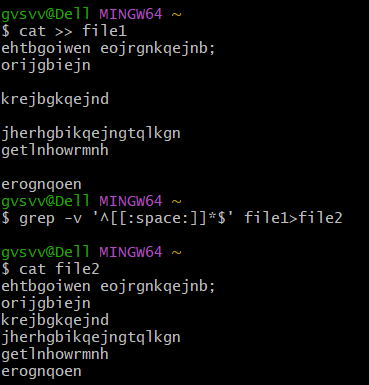
1i<HTML>: Inserts the tag <HTML> at the beginning of the file. The 1 specifies the

line number where the text should be inserted, and i indicates that the text

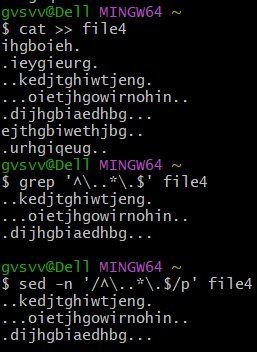
should be inserted before the specified line.

-e is used to specify multiple commands to sed.$a</HTML> tells sed to append the string </HTML> at the end of the file

10.21



10.22



10.23

grep '^.\{101,149\}$' filename

sed -n '/^.\{101,149\}$/p' filename

10.24

grep -E '\b(\w)\1{2}\b' filename

sed -n 's/\b\(\w\)\1\{2\}\b/&\n/gp' filename

Exercise-5

10.25

grep -o 'ENCRYPTION' foo | wc -l

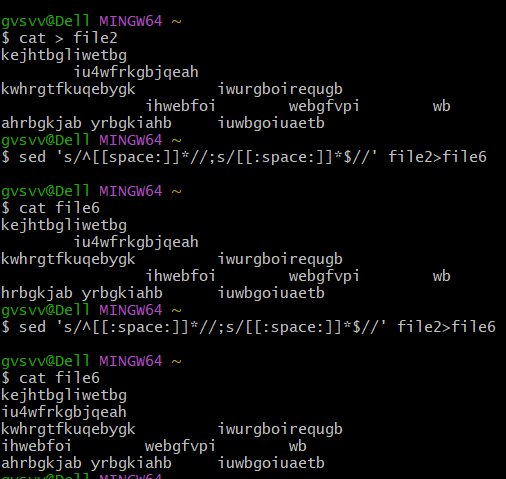
sed 's/ENCRYPTION/&\n/g' foo | grep -c 'ENCRYPTION'

10.26

sed -i '/^#/!b;/^#\!\/bin\/ksh/p;d' file.txt

sed -i '/^\/\\*/,/\\*\//d' file.txt

10.27



10.28

(i)

This sed command replaces occurrences of the word print with printf and

then replaces occurrences of the word printf with print in the file foo.

However, this command has a problem of replacing all occurrences of the

word print with printf, and then replacing all occurrences of the word printf

with print. As a result, if a line in the file already contains the word printf,

that word will be replaced with print first and then the replacement will be

undone in the next step, resulting in no net change.

(ii)

This sed command replaces occurrences of the word compute with calculate

and then replaces occurrences of the word computer with host in the file

foo.There is nothing technically wrong with this command, but it might not

always be the desired behavior, as it replaces all occurrences of the words

compute and computer regardless of their context.

10.29

sed -i 's/<B>/&STRONG>/;s/<\/B>/&\/STRONG>/' file.html

10.31

ls | sed 's/.\*\.//' | sort | uniq -c

1.ls lists all files in the current directory.

2.sed 's/.\*\.//' removes all characters before the last dot . in each filename.

This leaves only the extension.

3.sort sorts the extensions alphabetically.

4.uniq -c counts the number of occurrences of each unique extension and

displays the count along with the extension.

10.32

sed 's/^$//;/^$/d' input\_file | sort | awk 'NF{print $0"\n\n"}'

sed 's/^$//;/^$/d' input\_file removes any blank lines in the input file. The

^$ pattern matches empty lines, and the d command deletes them. This is

done to ensure that sort doesn't get confused by the empty lines.

sort- sorts the lines of the input file

awk 'NF{print $0"\n\n"}' reads each line of the sorted output and adds two

newlines to each non-blank line. This is done to restore the original

double-spacing of the file.