Experiment 6

6B

Theory

1.awk

The awk command is a Linux tool and programming language that allows users to process and manipulate data and produce formatted reports. The tool supports various operations for advanced text processing and facilitates expressing complex data selections.

awk allows users to perform various operations on an input file or text. Some of the available operations are:

* Scan a file line by line.
* Split the input line/file into fields.
* Compare the input line or fields with the specified pattern(s).
* Perform various actions on the matched lines.
* Format the output lines.
* Perform arithmetic and string operations.
* Use control flow and loops on output.
* Transform the files and data according to a specified structure.
* Generate formatted reports.

Inserting a pattern in front of an action in awk acts as a selector. The selector determines whether to perform an action or not. The following expressions can serve as patterns:

* Regular expressions.
* Arithmetic relational expressions.
* String-valued expressions.
* Arbitrary Boolean combinations of the expressions above.

A BEGIN rule is executed once before any text processing starts. In fact, it’s executed before awk even reads any text. An END rule is executed after all processing has completed. You can have multiple BEGIN and END rules, and they’ll execute in order.

If you want awk to work with text that doesn’t use whitespace to separate fields, you have to tell it which character the text uses as the field separator. For example, the /etc/passwd file uses a colon (:) to separate fields.

2.perl

"Perl" officially stands for "Practical Extraction and Report Language." It was originally a language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It quickly became a good language for many system management tasks. Over the years, Perl has grown into a general-purpose programming language. It's widely used for everything from quick "one-liners" to full-scale application development.

The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal). It combines some of the best features of sed, awk, and sh, making it familiar and easy to use for Unix users to whip up quick solutions to annoying problems. Its general-purpose programming facilities support procedural, functional, and object-oriented programming paradigms, making Perl a comfortable language for major projects.

Perl's roots in text processing haven't been forgotten over the years. It still boasts some of the most powerful regular expressions to be found anywhere, and its support for Unicode text is world-class. It handles all kinds of structured text, too, through an extensive collection of extensions. Those libraries, collected in the CPAN, provide ready-made solutions to an astounding array of problems.

Running Perl

The normal way to run a Perl program is by making it directly executable, or else by passing the name of the source file as an argument on the command line. (An interactive Perl environment is also possible.) Upon startup, Perl looks for your program in one of the following places:

* Specified line by line via -e or -E switches on the command line.
* Contained in the file specified by the first file name on the command line. (Note that systems supporting the #! notation, such as bash, invoke interpreters this way.)
* Passed in implicitly via standard input. This works only if there are no file name arguments--to pass arguments to a STDIN-read program you must explicitly specify a "-" for the program name.

Experiment no 6

6A

Theory

1.grep

On Unix-like operating systems, the grep command processes text line by line, and prints any lines which match a specified pattern.

Grep, which stands for "global regular expression print," is a powerful tool for matching a regular expression against text in a file, multiple files, or a stream of input. It searches for the PATTERN of text you specified on the command line, and outputs the results for you.

To search through nested directories and subdirectories, use the -r (recursive) option. Note that you don’t provide a file name on the command line, you must provide a path. Here we’re searching in the current directory “.” and any subdirectories:

2.egrep

The egrep command searches an input file (standard input by default) for lines matching a pattern specified by the Pattern parameter. These patterns are full regular expressions as in the ed command (except for the \ (backslash) and \\ (double backslash)). The following rules also apply to the egrep command:

* A regular expression followed by a + (plus sign) matches one or more occurrences of the regular expression.
* A regular expression followed by a ? (question mark) matches zero or one occurrence of the regular expression.
* Multiple regular expressions separated by a | (vertical bar) or by a new-line character match strings that are matched by any of the regular expressions.
* A regular expression may be enclosed in ( ) (parentheses) for grouping.

The new-line character will not be matched by the regular expressions.

The order of precedence for operators is [, ], \*, ?, +, concatenation, | and the new-line character.

3.sed

The sed stream editor performs basic text transformations on an input stream (a file, or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

If you do not specify INPUTFILE, or if INPUTFILE is "-", sed filters the contents of the standard input. The script is actually the first non-option parameter, which sed specially considers a script and not an input file if and only if none of the other options specifies a script to be executed (that is, if neither of the -e and -f options is specified).

A sed program consists of one or more sed commands, passed in by one or more of the -e, -f, --expression, and --file options, or the first non-option argument if none of these options are used. This documentation frequently refers to "the" sed script; this should be understood to mean the in-order catenation of all of the scripts and script-files passed in.

Commands within a script or script-file can be separated by semicolons (";") or newlines (ASCII code 10). Some commands, due to their syntax, cannot be followed by semicolons working as command separators and thus should be terminated with newlines or be placed at the end of a script or script-file. Commands can also be preceded with optional non-significant whitespace characters.

Each sed command consists of an optional address or address range (for instance, line numbers specifying what part of the file to operate on; see selecting lines for details), followed by a one-character command name and any additional command-specific code.