Лабораторна робота № 2

Створення командних файлів в Linux у вигляді Perl-скриптів

Мета: оволодіти навичками написання Perl-скриптів в Linux.

Завдання.

Скласти блок-схему алгоритму роботи Perl-скрипта.

Скласти програму на мові Perl згідно з варіантом завдання.

Вивести результати на екран.

Варіанти.

Написати Perl-скрипт, який вилучає каталог, ім’я якого передається через командний рядок. Якщо скрипт запустили без параметрів, то передбачити видачу сповіщення про правильний синтаксис його виклику. (Вказівка: спочатку вилучити всі файли з каталогів, вкладених в даний каталог (якщо такі каталоги є)).

Написати Perl-скрипт копіювання одного файла в другий. Передбачити введення імен файлів як з командного рядка, так і з екрану монітора.

Написати Perl-скрипт копіювання вмісту одного каталогу в інший каталог. Передбачити введення імен каталогів як з командного рядка, так і з екрану монітора.

Написати Perl-скрипт читання рядка текстового файлу з заданим номером. Передбачити випадки, коли номер заданого рядка більший за число рядків в файлі. Якщо номер рядка – від’ємне число, то прочитати всі рядки, починаючи з рядка з номером, рівним абсолютному значенню введеного від’ємного числа.

Написати Perl-скрипт, який замінює всі файли з розширенням *.mtx* в заданій директорії ОС Linux на файли з розширенням *.txt*. Файлів з розширенням *.xtx* в заданій директорії повинно бути не менше 5.

Написати Perl-скрипт, який перевіряє наявність файлу *index.htm* в директорії з *HTML*-файлами і при його наявності видає його вміст з допомогою браузера на екран. Тобто треба його повністю зчитати і вивести на екран.

Написати Perl-скрипт, який створює, перейменовує і вилучає файл *stroki.txt.*

Написати Perl-скрипт, який виводить вміст файлу *data.txt* на екран у вигляді *html*-файлу: (Вказівка. Для виведення даних з Perl-скрипту у виді html-файла треба вказати на це браузеру таким рядком*: print "Content-type: text/html\n\n*"; Починаючи з нього браузер будує *html*-документ зі змісту команди *print.* Отже, додавання в perl-скрипт рядка *print "<html><head>"* додає *html*-документ, що формується, рядок *<html><head>).*

Написати Perl-скрипт, який здійснює запит і введення імені користувача, порівняння з поточним логічним ім'ям користувача і виведення сповіщення: правильно/неправильно.

Написати Perl-скрипт, який переходить в інший каталог, формує файл з лістингом каталогу і повертається у вихідний каталог.

Написати Perl-скрипт, який виводить все відомості про вказаний файл або сповіщення про помилку, якщо файл не знайдено. Ім'я файлу передається з допомогою форми.

Написати Perl-скрипт, який виводить вказаний рядок файлу з вказаним зміщенням, тобто починає виведення не з початку. Використовуйте функції Seek і Getc для написання скрипта.

Написати Perl-скрипт, який створює в директорії з HTML документами HTML- файл, в якому записано таблицю Піфагора (10x10).

Написати Perl-скрипт, який виводить на екран список всіх файлів заданого каталогу, до яких більше 30 днів ніхто не звертався.

Написати Perl-скрипт, який вилучає всі файли заданого каталогу, до яких більше 30 днів ніхто не звертався.

Теоретична частина

ОС Linux має вбудовану мову програмування Perl (Practical Extraction and Report Language). Perl безкоштовна, структурована, різностороння і гнучка мова програмування, на якій написано більшість скриптів для Web. Perl інтерпретована, тобто Perl-програми не треба компілювати. Perl має широкий спектр інструментів для роботи з файлами.

На Perl знак # (дієз) означає коментар (ігнорується текст після дієзу до кінця рядка).

Перший рядок (#!*c:/usr/bin/perl або #!c:/usr/local/bin/perl*) будь-якого Perl-скрипта дає команду Linux запуску скрипта інтерпретатором Perl,

інстальованим за адресою *c:/usr/bin/perl* чи *c:/usr/local/bin/perl*. Цей перший рядок є коментар, але без нього скрипт може не спрацювати.

Застосування PERL-дебагера. В ньому легко по рядкам перевіряти роботу скриптів. Perl-дебагер вбудовано в Perl і запускається з командного рядка так: *C:\PERL> Perl -d hello.pl <ENTER>*. Perl завантажить скрипт *hello.pl* почне відлагодження. В Linux також можна написати рядок *#!/usr/bin/perl -d* в самому початку Perl-скрипта. Загрузка Perl-дебагера без загрузки скрипта: *C: \> Perl -de 0 <ENTER>* (ключ *-d* вказує Perl запустити дебагер, а аргумент *-e 0* - виконати Perl-скрипт з 0 рядків). При нормальній роботі Perl-дебагера на екран виводиться: *Loading DB routines from $RCSfile: perldb.pl,v $$Revision: 4.0.1.3 $$Date: 04/09/09 13:43:57 $ Emacs support available. Enter h for help. main '(p1000159:1): DB<1>*. Якщо виводиться *"Can't locate perldb.pl @INC"*, то Perl треба перевстановити. Команди дебагера: - *h* – виведення списку команд дебагера, - *n* – виконувати до наступного виразу; - *<CR>* - повторити останню команду *n* або *s*, - *p* вираз – друкувати вираз, - *q* – закінчити роботу, - *r* – виконувати до виходу з підпрограми, - *s* – крок у скрипті (з входом в підпрограму).

Приклади виведення на екран сповіщення *Hello Friends*:

1) `DB<1> prinf "Hello Friends\n"; <ENTER> Hello Friends DB<2> ;`

2) `<ENTER> DB<2> p "Hello Friends\n" <ENTER> Hello Friends DB<3>.`

Для введення кількох рядків в дебагер в кінці кожного рядка вставляють символ продовження рядка *</>: DB<3> for ($i = 0; $i < 10; $i++){\<ENTER>cont: print $i;\<ENTER> cont: } <ENTER> 0123456789 DB<4>*.

Скалярні змінні. Тип числових або символьних даних вказують символом$: $ім'я_скаляра = 4; (скалярній змінній $ім'я_скаляра присвоєно число 4). Скалярні змінні можна додавати і віднімати.

Список – це впорядковані скаляри, записані через кому в круглих дужках*:(5,$x,12,2+$y).*

Масив – це змінна, яка містить список. Перед змінною-масивом ставиться знак @:

@ім'я_масиву = *(5, $x, 12, 2+$y);*.

Доступ до елемента масиву отримують так: *$ім'я_масиву [номер_скаляра];* Номер_скаляра відраховується з нуля. Отже, звернення до змінної *$x* в цьому масиві таке: *$ім'я*_масиву [1];

Хеш – це асоціативний масив. Кожний елемент хеша має свій "ключ" доступності. Назва хешу починається з символу %. Додають новий елемент в хеш так: *$хеш {"ключ"} = "значення"*. Тепер %хеш містить "значення". Видобувають елемент "значення" так: $хеш {"ключ"}.

Керуючі структури. Всюди далі скорочення «оп» означає «оператор». Нижче наведено синтаксис керуючих Perl-структур:

Керуюча структура if/else:
if (умова_1) {
оп_1; оп_2; … # ці оператори виконуються, # якщо умова_1 вірна
} else {
 	
оп_3; оп_4; … # ці оператори виконуються,
#якщо умова_1 хибна
}
	Керуюча структура for:
	for (вираз_1; умова_1; вираз_2) {
	оп_1; оп_2; ...
	}
	# вираз_1 виконується 1 раз на початку циклу
	# блок операторів виконується, поки #умова_1 вірна
	# вираз_2 виконується в кінці кожного #циклу.

Керуюча структура while:
while (вираз_1) {
оп_1; оп_2; ...
}
# блок операторів виконується, 	
# поки вираз_1 вірний. 	
	Керуюча структура until:
	until (вираз_1) {
	оп_1; оп_2;
	}
	# блок операторів виконується,
	# поки вираз_1 хибний.

Керуюча структура do/while: do {оп_1; оп_2; } while вираз_1; # блок операторів виконується 1 раз. # далі блок виконується, поки вираз_1 вірний.	***Керуюча структура do/until:*** do {оп_1;оп_2; } until вираз_1; # блок операторів виконується 1 раз. # далі блок виконується поки вираз_1 хибний.
Керуюча структура foreach: foreach $x (@list) {оп_1; оп_2; ...} скалярній змінній $x по черзі присвоюються #значення зі списку @list. З кожним наступним значенням $x виконується #блок операторів.	

Функції користувача. Свою функцію пишуть в кінці скрипта. Її синтаксис такий:

sub ім'я_функції {оп_1; оп_2; ...} . Викликається вона вставкою в скрипт рядка: і'мя_функції();

Зарезервовані змінні

Для використання в Perl довгих імен змінних в заголовку скрипта пишуть *use English;*

В Perl є змінні лише для читання (значення в них не записати не можна):

**$_** – в неї за умовчанням можна вводити, присвоювати, заносити результат пошуку за заданим зразком: *while($_= <>){…}* ідентичне за дією *while(<>){…}.*

**$<digit>** – доступна лише для читання, як і змінні *$&, $`, $' і $+*.

$. – містить номер останнього прочитаного рядка з останнього прочитаного файлу. Доступна лише для читання.

$/ – містить символ розділення записів, що вводяться. За замовчуванням містить символ переведення рядка.

$| – містить 0 за умовчанням. Якщо в ній не 0, очищаються буфери кожний раз після виведення на друк, екран і т.п..

$, – містить символ-розділювач полів для оператора друку.

**$** – містить символ-розділювач записів для оператору друку. Задають *$* замість набирання *n* в кінці друку.

$" – подібна $, використовується при зверненні до списку величин в подвійних лапках. За замовчуванням містить символ пробіл.

**$;** – містить символ-розділювач для емуляції багатовимірних хешів. При зверненні до елементу хеша як *$toy{$a,$b,$c},* реально виконається *$toy{join($;,$a,$b,$c)}.* Не плутати з *@toy{$a,$b,$c},* що таке ж за дією, як *($toy{$a},$toy{$b},$toy{$c}).* За замовчуванням містить значення \034.

$# – формат для друку чисел (містить початкове значення %.20g).

$% – містить номер поточної сторінки, що виводиться.

$= – містить число рядків для друку поточної сторінки (звичайно містить значення 60).

$- – містить число ще не надрукованих рядків сторінки для каналу виведення при друкуванні.

$~ – містить ім’я поточного формату сповіщень (звичайно – ім’я дескриптора файлу).

$^ – містить ім’я поточного формату заголовку сторінки (звичайно – ім’я дескриптора файлу з _TOP в кінці).

$: – містить символи, після яких виведення рядка починається з нового рядка.

$! – якщо це змінна-число, то містить поточне значення *errno* (номер помилки). Якщо це змінна-рядок, то містить відповідне системне сповіщення про помилку.

$@ – містить сповіщення про синтаксичну помилку при виконанні останньої команди *eval()*. Сповіщення в ній не накопичуються. При виконанні *eval()* без помилок змінна містить 0.

$$ – містить ідентифікатор поточного процесу.

$< – містить ідентифікатор користувача (UID), якому належить поточний процес.

$> – містить ефективний UID поточного процесу.

$(– містить ідентифікатор групи (GID) користувача, якому належить поточний процес.

$) – містить ефективний GID поточного процесу.

$0 – містить ім’я файлу скрипта.

$ARGV – містить ім’я поточного файлу, з якого відбувається читання.

@ARGV – містить масив аргументів командного рядка, які були передані скрипту.

Наприклад, з Perl-скрипта можна отримати доступ до аргументів командного рядка шляхом запису в Perl-скрипті аргументів командного рядка в списковy змінну *@ARGV*. Для виведення аргументів командного рядка на екран: *while ($arg = shift @ARGV) { print "$arg\n"; })*

@INC - містить список точок входу в скрипт з конструкціями *do EXPR, require* та *use.*

%INC - містить входи для кожного файлу, який включається з допомогою операторів *do* або *require*. Ключами є імена файлів, а значеннями місця їх розміщення.

%ENV – містить поточне оточення процесу. Зміною вмісту хеша можна змінити оточення породженого (дочірнього) процесу.

%SIG – цей хеш використовується для встановлення обробників різних сигналів.

Дескриптори файлів

Доступ до файлу дає дескриптор – символьне ім’я в Perl-скрипті для посилання на файл, пристрій, сокет, програмний канал (дескриптор програмного каналу зв’язує процеси). Ім’я дескриптора файлу пишеться для зручності пошуку в скрипті лише великими буквами і не може бути ім’ям

зарезервованої Perl-функції. Одному дескриптору може відповідати лише один файл, хоч в різних місцях програми це може бути кожен раз інший файл.

Дескриптор файлу в Perl і в Linux – це два різних дескриптори. Дескриптор файлу в Linux – це 32-бітна адреса системної області для посилань на відкритий файл. Ця область містить інформацію про цей файл. Дескриптор файлу в Perl – це не адреса, а ім'я, придумане програмістом для посилань на файл. Отримати дескриптор файлу в Linux з Perl-програми можна командою *f ileno().*

Створює дескриптор файлу функція *open()* з двома параметрами – ім'ям дескриптора і ім'ям файлу з вказаним режимом доступу: *open(DESCRYPTOR, "> /temp/myfile.txt");* #open створює дескриптор *DESCRYPTOR,* приєднує його до файлу */temp/myfile.txt* і відкриває файл для запису (на це вказує символ ">"). Приклад запису в цей файл значення змінної *var*: *print DESCRYPTOR $var*;. Дескриптор не можна присвоїти, зберегти в змінній або передати як параметр в функцію, оскільки дескриптор – не змінна (його ім'я не містить префікса змінної *Perl ($, @ або %)*). Для цих дій перед ім'ям дескриптора пишуть префікс * – посилання на глобальний тип даних. Наприклад, записати в файл значення змінної можна ще й так*:$name1 = *DESCRYPTOR; print $name1 $var*; #змінна *$name1* заміщує *DESCRYPTOR,* в який виводиться значення змінної *$var*.

будь-якому Perl-скрипті завжди є три наперед заданих дескриптори (*STDIN, STDOUT и STDERR*), де *STDIN* зв'язаний з клавіатурою, *STDOUT і STDERR* – з екраном монітора. *STDIN* використовує функція o, якщо в командному рядку виклику Perl-скрипта немає списку файлів. STDOUT за умовчанням використовують функції print і die, а STDERR – функція *warn*. Можна перенаправляти стандартне введення і виведення в інші файли дописуванням префіксів *'>'*(в файл) і *'<'*(з файлу): *perl script1.pl <in.dat >out.dat*; – при виконанні *script1.pl* дані з файлу *in.dat* запишуться в файл *out.dat* без показу на екрані. Перенаправлення з Perl-скрипта можливе у виді: open(*STDIN, "in.dat"); open*(*STDOUT, ">out.dat"*); *open(STDERR, ">err.dat");*
Тепер все стандартне введення/виведення йтиме крізь вказані в open() файли. Таке перенаправлення можна робити в Perl-скрипті лише раз (функція open() не повертає початкові призначення дескрипторів *STDIN, STDOUT і STDERR).*

Вибрані вбудовані Perl-функції для роботи з файлами і каталогами
binmode FILEDESCR – для читання або запису в файл за його дескриптором в бінарному режимі.

caller EXPR – видає контекст поточного виклику підпрограми. Якщо *EXPR* скаляр, то видає *TRUE* (при знаходженні функції в тілі підпрограми), *eval()* або *require(),* інакше – *FALSE*. Якщо *EXPR* список, то видає: *($package, $filename,$line)=caller*. З аргументом *EXPR* видає інформацію для використання дебагером при друкуванні карти стеку. Значення *EXPR* відмічає глибину стеку до поточного запису: ($package, $filename, $line, $subroutine, $hasargs, $wantargs) = caller($i);

chdir EXPR – змінює поточну директорію на вказану в *EXPR*, якщо остання існує. Без *EXPR* поточною стає домашня директорія. Видає *TRUE* в разі успіху, інакше - *FALSE*.

chmod LIST - змінює права доступу до файлів зі списку *LIST*. Першим аргументом списку є цифрова (за звичай вісімкова) маска доступу. Видає число файлів з успішно зміненими правами доступу. Наприклад: *$cnt = chmod 0700 'loo','tar'; chmod 700 @executables; chmod 0666 'f1', 'f2', 'f3';*

chown LIST – змінює власника або групу, якій належить список файлів. Першими двома аргументами завжди є *uid* та *gid*. Видає число успішних змін.

close FILEDESCR – закриває файл з дескриптором *FILEDESCR*. Наприклад:

open(FD5 '/usr/home/stud'); ... close FD5;

closedir DIRDESCR – закриває каталог, відкритий функцією *opendir*.

dbmopen ASSOC, DBNAME, MODE – Зв'язує *dbm(3)* або *ndbm(3)* файл з асоціативним масивом. *ASSOC* – ім'я асоціативного масиву. *DBNAME* – ім'я бази даних (без *.dir* або *.pag* розширення). Якщо база даних не існує, вона створюється з правами доступу, вказаними в *MODE*: *dbmopen(%HIST,'/usr/lib/news/history', 0600); while (($key, $val) = each %HIST){ print $key, '=', unpack('L',$val),\n;} dbmclose(%HIST);*

dbmclose ASSOC – розриває зв'язок між файлом і асоціативним масивом.

die LIST – за межами *eval()* друкує значення *LIST* в *STDERR* і виходить з програми з поточним значенням *$!*. Якщо в *$!* нуль, то приймає значення *$? >> 8*, а якщо значення *$?>>8* нульове, то *255*. Всередині *eval()* сповіщення про помилку заноситься в змінну *$@* і *eval()* переривається з невизначеним значенням. Приклад:

open(FL, "/root/rm-rf") || die "Can't open file.\n";

do EXPR – вважає *EXPR* ім'ям файлу і запускає вміст цього файлу як програму на *Perl*. Часто нею включають в скрипт бібліотечні підпрограми.

Наприклад: *do 'text.pl'*; ідентичне: *eval 'cat text.pl'*;. Підключати бібліотечні модулі зручніше функціями *use* та *require*.

eof FILEDESCR – видає 1, якщо наступне зчитування видає кінець файлу або якщо *FILEDESCR* не було відкрито. Без аргументу *eof* опрацьовує останній файл, що зчитувався. Функція мало використовується, оскільки в *Perl* оператори читання видають невизначене значення в кінці файлу.

getc FILEDESCR – читає і видає наступний символ з файлу читання, приєднаного до *FILEDESCR* або пустий рядок у випадку кінця файлу. Без *FILEDESCR* зчитує з *STDIN*.

Код нижче читає і виводить 15 байт з файлу *"file.txt": open (file,"file.txt"); while ($u<=15){ print getc(file);} continue{$u++} close file*;

link OLDFILE,NEWFILE – створює файл *NEWFILE*, приєднаний до файлу *OLDFILE*. В *UNIX* для одного файлу можна створити кілька імен. Видає 1 в разі успіху і 0, якщо інакше.

mkdir FILENAME,MODE – створює директорію з іменем *FILENAME* і правами доступу, вказаними в змінній *MODE* (вісімкове число 0777). В разі успіху видає 1, інакше видає 0 і встановлює значення змінної *$!(errno)*. Якщо задано ім’я каталогу без шляху до нього, то він створюється в поточному каталозі.

open FILEDESCR,EXPR – відкриває файл з іменем *EXPR* і ставить йому у відповідність файлову змінну *FILEDESCR* (дескриптор файлу). Без *EXPR* змінна з іменем *FILEDESCR* містить ім’я файлу. Спецсимвол (<,>,>>,+<,+>) перед ім’ям файлу визначає режим доступу до файлу при його відкритті. Наприклад: *open (FILEDESCR,"EXPR");* або *open (FILEDESCR,"<EXPR");* – відкриває файл тільки для читання (файл має існувати);

open (FILEDESCR,">EXPR"); – відкриває файл для запису в його початок зі знищенням попереднього вмісту файлу (якщо файл не існує, то створюється);

open (FILEDESCR, ">>EXPR"); – відкриває файл для дозапису в його кінець без знищення попереднього вмісту файлу (якщо файл не існує, то створюється);

open (FILEDESCR,"+<EXPR"); – відкриває файл для читання і запису без знищення попереднього вмісту файлу;

open (FILEDESCR, "+>EXPR"); – створює файл для читання і запису зі знищенням попереднього вмісту файлу;

open (FILEDESCR,"+>>EXPR"); – створює файл для читання і дозапису кінець без знищення попереднього вмісту файлу;. По закінченні роботи з файлом завжди пишуть *close (FILEDESCR);* (закривається файл *FILEDESCR*).

Спецсимволом також може бути знак ”|”, який може ставитися перед ім’ям або відразу ж після імені файлу. Знак ”|” перед ім’ям файлу означає, що воно є командою створення програмного каналу – вихід з *FILEDESCR* передасться на вхід *EXPR*. Наприклад: *open (FILEDESCR,"|EXPR");* – направити інформацію на вхід програми;. Знак ”|” після імені файлу означає, що вихід з *EXPR* передасться на вхід *FILEDESCR*, тобто з *FILEDESCR* відбуватиметься читання. Наприклад: *open (FILEDESCR,"EXPR|");* – зчитати інформацію з виходу програми. Фрагмент програми показує використання функцій *open* і *close*:

```
    open(InFile, " proba.dat") || die; # відкриваємо для читання proba.dat
      open(OutFile, ">proba.dat") || die;    # створюємо proba.dat
      $AuxFile = ">>proba.dat"; open(Aux, $AuxFile) || die; # відкриваємо для дозапису proba.dat
close(InFile); close(OutFile); close(Aux);
```

open DIRDESCR,EXPR – відкриває директорію з ім’ям *EXPR*, видає *TRUE* в разі успіху.

opendir DIRDESCR,DIRNAME – відкриває директорію з ім’ям DIRNAME. Приклад Perl-скрипта, який перевіряє, чи є всі файли каталогу двійковими (без перевірки вмісту вкладених каталогів): *#! perl –w opendir FDIR, "/usr/prog"; while ($name=readdir FDIR) {next if -d $name; # каталог print("/usr/prog/$name: двійковий\n") if -b $name; #Двійковий файл } closedir FDIR*;

print FILEDESCR,LIST – друкує рядок або кілька рядків, розділених комою. *FILEDESCR* може бути ім’ям скалярної змінної, яка містить дескриптор файлу. Без цієї змінної друк йде в вибраний канал виводу. Без змінної *LIST* друкує змінну *$_* в *STDOUT*.

printf FILEDESCR, LIST – еквівалент print *FILEDESCR, sprintf (LIST)*. Перший аргумент *LIST* інтерпретується як формат друку.

read FILEDESCR,SCALAR,LENGTH,OFFSET – зчитує *LENGTH* байт даних з *FILEDESCR* в змінну *SCALAR*. Видає число зчитаних байт або невизначеність в разі помилки. Для зчитування даних не з початку рядка вводять значення змінної *OFFSET*.

Положення вказівника зберігається. Код нижче читає і виводить 15 байт з файлу "file.txt": *open (file,"file.txt"); read("file",$u,5); print $u,"
"; read("file",$u,10); print $u; close file;*

readdir DIRDESCR, DIRNAME – для відкритого каталогу видає список імен всіх файлів каталогу. В скрипті допускаються дескриптори файлу і каталогу з однаковими іменами: *open FF, "/usr/out.dat" # дескриптор файлу opendir FF, Rusr" # дескриптор каталогу*. Приклад використання функції *readdir* для виведення на екран списку файлів з поточного каталогу: *opendir(Dir, $INC[2]) || die; while ($file = readdir(Dir)) { print "$file \n" } closedir(Dir);*, де змінна *$INC[2]* дає доступ до поточного каталогу. Змінюючи *$INC[2]* на *$ARGV[0]*, скрипт виводить на екран список файлів з каталогу, вказаному в командному рядку.

readlink EXPR – видає значення символьного посилання, якщо посилання існує. Якщо посилання не існує, то видає *fatal error* і встановлює значення змінної *$!*. За умовчанням опрацьовує змінну *$_*.

require EXPR – підключає модулі. Приклад: *require "oraperl.pm";*

rename (старе_ім’я, нове_ім’я_файлу); – перейменовує файл (видає 1 в разі успіху, інакше - 0).

rewinddir DIRDESCR – поточну позицію в каталозі встановлює на початок (можна повторно читати імена файлів каталогу, не закриваючи його). Єдиним параметром цієї функції є дескриптор відкритого каталогу.

rm FILENAME – вилучає файл або директорію з заданим ім’ям. Видає 1 в разі успіху, інакше - 0 і встановлює значення змінної *$!*. За замовчуванням опрацьовує аргумент *$_*.

rmdir DIRNAME – вилучає пустий каталог (без вкладених пустих каталогів). Якщо параметру не задано, то вставляють спеціальну змінну *$_*, яка містить значення «істина» при успішному знищенні каталогу і пояснення помилки при неможливості знищення.

seek FILEDESCR, POSITION, WHENCE – встановлює курсор в файлі, визначеному в змінній *FILEDESCR*, на позицію *POSITION* в режимі, вказаному змінній *WHENCE* (або відлік). Якщо в змінній *WHENCE* 0, то позиція починає відлік від початку файлу, якщо 1 – від поточної позиції і якщо 2, то від кінця файлу. Видає 1 в разі успіху і 0 – інакше.

select FILEDESCR – видає поточний вибраний *FILEDESCR*. Направляє виведення в *FILEDESCR*.

stat FILENAME – видає 13 - елементний масив параметрів вказаного файлу або пустий список в разі помилки. Наприклад: *($dev,$ino,$mode,$nlink,$uid,$gid,$rdev,$size, $atime,$mtime,$ctime,$blksize,$blocks) = stat($filename)*; *$dev* – ім'я пристрою ;*$ino* - номер і-вузла; *$mode* - права доступу; *$nlink* - число зв'язків; *$uid* - ідентифікатор власника; $gid - ідентифікатор групи; *$rdev* - тип пристрою; *$size* розмір файлу в байтах; *$atime* - дата останнього звернення; *$mtime* - дата останньої модифікації; *$ctime* - дата останньої зміни статусу; *$blksize* - розмір блока на диску; *$blocks* - число блоків в файлі. Код виведення розміру файлу в байтах: *$u = (stat("file.txt"))[7];print $u;}.*

sysopen ДЕСКРИПТОР, ІМ'Я, ПРАПОР [, ДОЗВІЛ]; – може, крім завдання режиму відкриття файлу, задавати права доступу до файлу: режим відкриття файлу задає ПРАПОР – число як результат побітового *АБО* (|) над константами режимів з модуля Fcntl. Константи режиму доступу до файлу: *O_RDONLY* (тільки читання), *O_WRONLY* (тільки запис), *O_RDWR* (читання і запис), *O_CREAT* (створення файлу, якщо він не існує), *O_EXCL* (завершення з помилкою, якщо файл вже створено), *O_APPEND* (додавання в кінець файлу %). Права доступу (параметр ДОЗВІЛ) задаються у вісімковій системі з урахуванням поточного значення маски доступу до процесу з функції *umask()*. Якщо цей параметр не задано, то Perl використовує значення *0666* для звичайних файлів. Для каталогів і виконуваних файлів використовують *0777*. Приклад операцій відкриття файлів функцією *open ()* та *sysopen ():*

```
use Fcntl; open FF, "< file.txt"; sysopen FF, "file.txt", O_RDONLY; # лише читання open FF, "> file.txt";sysopen FF, "file.txt", O_WRONLY | O_CREAT | O_TRUNC; # лише запис (створює новий файл або очищує існуючий)open FF, ">> file.txt";sysopen FF, "file.txt", O_WRONLY | O_CREAT | O_APPEND; # дозапис в кінець (створюється, якщо не існує)open FF, "+< file.txt"; sysopen FF, "file.txt", O_RDWR; # читання/запис існуючого файлу open FF, "+> file.txt"; sysopen FF, "file.txt", O_RDWR | 0_CREAT I O_TRUNC; # читання/запис (файл очищається)
```

tell FILEDESCR – видає поточну позицію курсору у відкритому файлі *FILEDESCR*. Без аргументу опрацьовує файл, який читався останнім. Наприклад, наступний код читає 15 байт з файлу "file.txt" (довжина файлу повинна бути більше 15 байт).

```
print "<pre>"; open (file, "file.txt"); while (tell(file) < 15){
print getc(file);} print "</pre>";
```

truncate FILEDESCR, LENGTH – обрізає файл *FILEDESCR* до заданої довжини.

Приклад запису в файл file.txt рядка "This is a sample file" і вкорочення його до 20 байт: `#!/usr/bin/perl print "Content-type: text/html\n\n"; $string = "This is a sample file"; print "Write to file: $string"; open (file,">file.txt");`

`print file $string; close file; truncate ("file.txt",20); open (file, "file.txt"); $string=<file>; close file; print "Read from file: ",$string;`

unlink LIST – вилучає список файлів і видає число успішно вилучених файлів. Не вилучає каталоги, якщо ви не *root*-користувач. Безпечніше використати функцію *rmdir*.

use Module LIST – приєднує модуль до програми: *use strict qw(subs,vars,refs);*

utime Date1,Date2,список_файлів – змінює дату звернення і модифікації файлів зі списку. Перші два елемента списку повинні вказувати нове значення дати звернення і модифікації. Видає число змінених файлів.

write – створює запис, який може складатися з кількох рядків у відповідному файлі, з використанням формату, що відповідає цьому файлу. Формат для поточного каналу виведення встановлюється присвоєнням змінній *$~{ }* назви формату.

-X – перевірка файлу на можливість дій з ним. Це унарний оператор з одним аргументом - або ім'ям файлу, або вказівником файлу. Перевіряє одну з умов. Без аргументу береться значення змінної *$_*. Аргумент можна писати в круглих дужках. *'X'* має такі значення:

-r – файл дозволено читати ефективним *uid/gid* ; -w – в файл дозволено записувати ефективним *uid/gid*; -x - файл дозволено виконувати ефективним *uid/gid*; -o - файл належить ефективному *uid* (ідентифікатор користувача); -R - файл дозволено читати реальним *uid/gid*; -W –в файл дозволено записувати реальним *uid/gid*; -X - файл дозволено виконувати реальним *uid/gid*; -O – файл належить реальному *uid*; *-e* - файл існує; *-z* – файл пустий; *-s* - файл не пустий; *-f* – звичайний текст; *-d* - директорія; *-l* - символічне посилання; *-p* - pipes (конвеєр); *-S* - socket (сокет); *-b* - спеціальний блочний пристрій*; -c* –

спеціальний символьний пристрій; *-t* – вказівник на пристрій *tty*; *-u* - встановлено біт *setuid*; *-g* - встановлено біт *setgid*; *-k* - встановлено біт *sticky*; *-T* - текстовий файл; *-B* – двійковий файл; *-M* - "вік" файлу в днях на момент старту скрипта; *-A* - днів з останнього читання; *-C* - днів з останньої модифікації *inode*.

Приклад перевірки наявності файлу "file.txt" в поточній директорії і виведення при наявності кількості днів з моменту останнього звернення до нього:

```
if (-e("file.txt")){print (-A("file.txt"))} else {print "Файл не створено"}.
```

Синтаксис регулярних виразів для обробки тексту

Регулярний вираз – це зразок, що складається з символів. Він використовується для аналізу скриптом вхідних даних з урахуванням пробілів, ком, символів табуляції та інших розділювачів. В регулярних виразах Perl для скороченого запису використовує символи:

Символ	Опис
.	Відповідає будь-якому символу (крім символу нового рядка)
(..)	Групує послідовність елементів
+	Задовольняє попередньому зразку один або більшу кількість раз
-	Задовольняє зразку нуль або один раз
*****	Відповідає зразку один або нуль раз
[...]	Відповідає символу з заданої множини
[^...]	Відповідає символу з множини, отриманої запереченням
(...|...|...)	Відповідає одній з альтернатив
^	Відповідає початку рядка
$	Відповідає зразку в кінці рядка
{n,m}	Відповідає зразку від n до m раз
{n}	Відповідає зразку точно n раз
{n,}	Відповідає зразку мінімум n раз
\n\t etc.	Відповідає знаку нової лінії, символу табуляції і т. д.
\b	Відповідає на межі слова
\B	Відповідає всередині меж слова
\d	Відповідає цифрі
\D	Відповідає не цифрі
\s	Відповідає пробілу
\S	Відповідає не пробілу
\w	Відповідає букві або цифрі
\W	Відповідає символу, що не є ні буквою, ні цифрою

Perl регулярні вирази (зразки, шаблони) обмежуються з обох боків слешами (похилими рисками), наприклад, в виді /pattern/.

Наприклад:

```
  наступні регулярні вирази істинні, якщо:
/ig/          #    рядок містить 'ig'
/(b|d|f)ig/      # рядок містить 'big', 'dig' або 'fig'
/[0-9]+/        # рядок містить число
/[A-Za-z][A-Za-a0-9_]*/ # рядок містить ідентифікатор
```

Детальніше з роботою з файлами в Perl можна ознайомитися на сайтах: *http:\\www.PERL.org.ru* або *http:\\www.perl.com*

Приклад. Створити Perl-скрипт, який виводить на екран дату-час, інформацію про вільне місце на дисках та про використання оперативної пам'яті комп'ютера.

Рішення. Створюємо в *Kword* або в *StarOffice* наступний текст програми і зберігаємо його з розширенням *.txt* або *.pl*:

```
use Term::ANSIColor; system clear; print color("magenta");
system date;
ptint color("white"), "\n"; system df; print color
("yellow"), "\nOperative memory\n"; system free; print
color("reset"),"\n";
```

При збереженні програми даємо ім'я файлу: *fds.txt*. Запускається Perl-скрипт на виконання з командного рядка Linux так: *[stud@localhost Perl] $ perl –w fds.txt.*

Контрольні запитання

Які сервіси надає типова операційна система?

Для ЕОМ якого покоління були розроблені перші ОС?

Як класифікуються ОС за режимом обробки задач?

Що таке Perl-скрипт?

Які керуючі структури є в Perl?

Які команди роботи з файлами і каталогами в Perl використано в Perl-скрипті зробленої практичної роботи?

Які регулярні вирази Perl використано в Perl-скрипті зробленої практичної роботи?