perl commands used:

1. perl -v : to know if perl is installed propelry and the version of perl

2. perldoc perl : docs of perl

3. perldoc -tf print : dc of the print command

4. run a perl script by perl <abc.pl>

5. in UNIX running the perl prg as a prg not a script do : chmod 755 <abc> now u can run

it as ./abc or abc. Don't ever name your program "test" under Unix.

The Unix shells have a command called test, and you'll be very frustrated when the

wrong program runs

6. scalar constants called literals: numeric literals and string literals.

7.Variables such as $\_, $", $/, $2, and $$ are special variables and should not be used

as normal variables in your Perl programs.

8.If your program has -w on the #! line at the beginning,

the directive "use warnings" appears in your program (Perl 5.6).If you invoke Perl with

the -w

switch on the command line,try to use the value of a variable that you haven't

previously set, Perl responds with this error message when your program runs and you try

to use the value: Use of uninitialized value. i.e : using : #!/usr/bin/perl -w

9.Operator Sample Usage Result

int int(5.6234) Returns the integer portion of its argument (5).

length length("nose") Returns the length of its string argument (4).

lc lc("ME TOO") Returns its argument shifted to lowercase letters ("me too").

uc uc("hal 9000") Returns its argument shifted to uppercase letters ("HAL 9000").

cos cos(50) Returns the cosine of 50 in radians (.964966).

rand rand(5) Returns a random number from 0 to less than its argument. If the

argument is omitted, a number between 0 and 1 is returned.

10. **autoincrement operator** (++). The autoincrement operator adds 1 to its operand:

$counter++;

$countdown=10;

$countdown--; # decrease to 9

11. Let me add a final note on the autoincrement operator: When the operator is applied to a text string, and the

text string starts with an alphabetic character and is followed by alphabetic characters or numbers, this

operator becomes magical. The last (rightmost) character of the string is increased. If it's an alphabetic

character, it becomes the next letter in sequence; if it's numeric, the number increases by 1. You can carry

More Operators

across alphabetic and numeric columns as follows:

$a="999";

$a++;

print $a; # prints 1000, as you'd expect

$a="c9";

$a++;

print $a; # prints d0. 9+1=10, carry 1 to the c.

$a="zzz";

$a++;

print $a; # prints "aaaa".

12. **Angle Operator (<>)**

The angle operator (<>),a diamond operator, is used for reading and writing files;

<STDIN>. This form indicates to Perl that a

line of input should be read from the standard input device—usually the keyboard. The <STDIN> expression

returns the line read from the keyboard:

print "What size is your shoe? ";

$size=<STDIN>;

print "Your shoe size is $size. Thank you!";

What is size is your shoe? **9.5**

Your shoe size is 9.5

. Thank you!

The <STDIN> expression reads from the keyboard until the user presses the Enter key. The entire line of

input is returned and is placed in $size. The line of text returned by <STDIN> also includes the newline

character that the user typed by pressing Enter. That's why the period and "Thank you!" appear on a new line

in the preceding display—the newline was part of the value of $size. Often, as here, you don't want the

newline character at the end of the string—just the text. To remove it, you can use the chomp function as

follows:

print "What size is your shoe?";

$size=<STDIN>;

chomp $size;

print "Your shoe size is $size. Thank you!\n";

chomp removes any trailing newline character at the end of its argument

13. One can write expressions like :

$a=$a+3;

can be reduced to

$a+=3;

14. Concatenation can be performed only with the concatenation operator (.).

a.

True

b.

False

b. False. A motto in the Perl community is "There Is More Than One Way To Do

It" (TIMTOWTDI). Concatenation can be performed by including two (or more) scalars in

a double-quoted string, as follows:

qq($a$b$c);

qq behaves in every way like a pair of double quotation marks. This means it can

interpolate variables.

15. if ( $r == 5 ) {

print 'The value of $r is equal to 5.';

}

The expression being tested is $r == 5. The == symbol is an equality operator.

16. if else function :

$r=<STDIN>;

chomp $r;

if ($r == 10) {

print '$r is 10', "\n";

} else {

print '$r is something other than 10...', "\n";

$r=10;

print '$r has been set to 10', "\n";

}

17. if (*expression1*) # If expression1 is true ...

*block1* # ...run this block of code.

elsif (*expression2*) # Otherwise, if expression2 is true...

*block2* # ...Run this block of code.

else

*block3* # If neither expression was true, run this.

Eg:

$r=10;

if ($r==10) {

print '$r is 10!';

} elsif ($r == 20) {

print '$r is 20!';

} else {

print '$r is neither 10 nor 20';

}

16. **Table 3.1. Numeric Relational Operators**

**Operator Example Explanation**

== $x == $y True if $x equals $y

> $x > $y True if $x is greater than $y

< $x < $y True if $x is less than $y

>= $x >= $y True if $x is greater than or equal to $y

<= $x <= $y True if $x is less than or equal to $y

!= $x != $y True if $x is not equal to $y

#!/usr/bin/perl -w

2:

3: $im\_thinking\_of=int(rand 10);

4: print "Pick a number:";

5: $guess=<STDIN>;

6: chomp $guess; # Don't forget to remove the newline!

7:

8: if ($guess>$im\_thinking\_of) {

9: print "You guessed too high!\n";

10: } elsif ($guess < $im\_thinking\_of) {

11: print "You guessed too low!\n";

12: } else {

13: print "You got it right!\n";

14: }

17. String comparision: The following is a fake comparision . How perl works over it is :

The two values $first and $last actually test *equal* to each other.

If

nonnumeric strings are used when Perl is expecting numeric values, the strings evaluate to zero. So the

preceding if expression looks something like this to Perl: if ( 0 == 0 ). This expression evaluates to

true, and that's probably not what you wanted.

**By the Way**

If warnings are turned on, trying to test two alphabetic values (simple and Simon in the

preceding snippet) with == will generate a warning message when the program runs to alert you

to this problem.

$first="Simon";

$last="simple";

if ($first == $last) { # == is not what you want!

print "The words are the same!\n";

}

So use : If you want to test nonnumeric values, you can use another set of Perl operators

**Table 3.2. Alphanumeric Relational Operators**

**Operator Example Explanation**

The if Statement

Eq $s eq $t True if $s is equal to $t

Gt $s gt $t True if $s is greater than $t

Lt $s lt $t True if $s is less than $t

Ge $s ge $t True if $s is greater than or equal to $t

Le $s le $t True if $s is less than or equal to $t

Ne $s ne $t True if $s is not equal to $t

These operators decide "greater than" and "less than" by examining each character left to right and comparing

them in ASCII order. This means that strings sort in ascending order: most punctuation first, and then

numbers, uppercase, and finally lowercase. For example, 1506 compares less than Happy, which compares

less than happy

#!/usr/bin/perl

use strict;

use warnings;

my $string1 = 'three';

my $string2 = 'five';

if ($string1 eq $string2) {

print "Equal\n";

} else {

print "Not equal\n";

}

Also :

|  |  |
| --- | --- |
| **Character** | **Description** |
| \L | Transform all letters to lowercase |
| \l | Transform the next letter to lowercase |
| \U | Transform all letters to uppercase |
| \u | Transform the next letter to uppercase |
| \n | Begin on a new line |
| \r | Applys a carriage return |
| \t | Applys a tab to the string |
| \f | Applys a formfedd to the string |
| \b | Backspace |
| \a | Bell |
| \e | Escapes the next character |
| \0nn | Creates Octal formatted numbers |
| \xnn | Creates Hexideciamal formatted numbers |
| \cX | Control characters, x may be any character |
| \Q | Do not match the pattern |
| \E | Ends \U, \L, or \Q functions |

Examples of regular expression comparision in strings:

The first example checks that a string does not contain any numbers:

#!/usr/bin/perl

use strict;

use warnings;

my $string1 = 'Jimmy has 17 dollars and 20 cents saved up';

if ($string1 !~ m/\d/) {

print "Doesn't contain numbers\n";

} else {

print "Does contain numbers\n";

}

exit 0;

And the next example shows how to check if a string contains a phrase like: '4 dollars and 25 cents':

#!/usr/bin/perl

use strict;

use warnings;

my $string1 = 'Jimmy has 17 dollars and 20 cents saved up';

if ($string1 =~

m/ # Match

\d+ # 1 or more digits

\sdollars\sand\s # ' dollars and '

\d+ # 1 or more digits

\scents # ' cents'

/ix # i = Ignore case

# x = allow the regexp to go over multiple lines

) {

print "Contains dollars and cents!\n";

}

exit 0;

use strict;  
use warnings;  
use Data::Dumper;  
  
#                  1    22     333  
my $message = 'this that  other   555';  
my @cellMessage = split /\s{2,}/, $message;  
print Dumper(\@cellMessage);  
  
\_\_END\_\_  
  
$VAR1 = [  
          'this that',  
          'other',  
          '555'  
        ];

@nd example :

@cellMessage = split(/\s{2,}/, $message);  
                foreach(@cellMessage){  
                    print "$\_ \n";  
                }

because you will match any whitespace character (tabs, spaces, etc...). The problem with your original code was that the split instruction is looking for a pattern and the regex you provided was resulting in the empty string //, which splits $message into individual characters.

The person who asked this question did this :

|  |  |
| --- | --- |
| **0**down vote[favorite](http://stackoverflow.com/questions/3366533/splitting-a-string-with-multiple-white-spaces-with-perl)  share [fb]share [tw] | I am trying to split a string with multiple white spaces. I only want to split where there are 2 or more white spaces. I have tried multiple things and I keep getting the same output which is that it splits after every letter. Here is the last thing I tried  @cellMessage = split(s/ {2,}//g, $message);                 foreach(@cellMessage){                     print "$\_ \n";                 } |

One example for reading a file:

Parsing Text Files

How to parse simple text files with Perl

From [Kirk Brown](http://perl.about.com/bio/Kirk-Brown-16393.htm), former About.com Guide

**Filed In:**

1. [Programming Perl](http://perl.about.com/od/programmingperl)
2. > [File System](http://perl.about.com/od/filesystem)

[**Ads**](javascript:zpu(512,uy+'/z/ad/wasl.htm',450,425,'wao',100,100))

[**Free CSV to Database Tool**](http://googleads.g.doubleclick.net/aclk?sa=l&ai=BbW3XwYnKTpK1N86JgALvwtCkBseLjccC1_zjtSKVyKa5A4C-kgEQARgBIKil9wEoCTgAUO_uyuj______wFgyf77iZikpBOgAb_K2vkDsgEOcGVybC5hYm91dC5jb23IAQHaATlodHRwOi8vcGVybC5hYm91dC5jb20vb2QvZmlsZXN5c3RlbS9hL3BlcmxfcGFyc2VfdGFicy5odG2AAgGoAwG4AwHoA-AF6APdBfUDAAQAAA&num=1&sig=AOD64_22q2VQyXGpd4wdZSWm4vr6yo-dAg&client=ca-primedia-premium_js&adurl=http://csvexpress.expressor-software.com/%3FcampaignID%3D70140000000MwlV%26keyword%3Dconvert%2520csv%2520to%2520database)Fast CSV to Database Conversion Get Free Tool to Load CSV Files Now[csvexpress.expressor-software.com](http://googleads.g.doubleclick.net/aclk?sa=l&ai=BbW3XwYnKTpK1N86JgALvwtCkBseLjccC1_zjtSKVyKa5A4C-kgEQARgBIKil9wEoCTgAUO_uyuj______wFgyf77iZikpBOgAb_K2vkDsgEOcGVybC5hYm91dC5jb23IAQHaATlodHRwOi8vcGVybC5hYm91dC5jb20vb2QvZmlsZXN5c3RlbS9hL3BlcmxfcGFyc2VfdGFicy5odG2AAgGoAwG4AwHoA-AF6APdBfUDAAQAAA&num=1&sig=AOD64_22q2VQyXGpd4wdZSWm4vr6yo-dAg&client=ca-primedia-premium_js&adurl=http://csvexpress.expressor-software.com/%3FcampaignID%3D70140000000MwlV%26keyword%3Dconvert%2520csv%2520to%2520database)

[**The Art Institutes**](http://googleads.g.doubleclick.net/aclk?sa=l&ai=BJX3-wYnKTpK1N86JgALvwtCkBqfOsvgBv5r0qh7AjbcBoL7DAhACGAIgqKX3ASgJOABQr6Sr7AJgyf77iZikpBOgAe2c-uoDsgEOcGVybC5hYm91dC5jb23IAQHaATlodHRwOi8vcGVybC5hYm91dC5jb20vb2QvZmlsZXN5c3RlbS9hL3BlcmxfcGFyc2VfdGFicy5odG2AAgHIApP2nxCoAwG4AwHoA-AF6APdBfUDAAQAAA&num=2&sig=AOD64_02HyAIIuKRPNdssj5kCIKQuLJ38g&client=ca-primedia-premium_js&adurl=http://www.the-art-institutes.info/index.php/media_arts/audio_video_film_production/%3Feng%3Dgoogle%26kw%3Dfilm%2520school)Get info on our Audio, Video & Film production classes. Many locations.[www.The-Art-Institutes.info](http://googleads.g.doubleclick.net/aclk?sa=l&ai=BJX3-wYnKTpK1N86JgALvwtCkBqfOsvgBv5r0qh7AjbcBoL7DAhACGAIgqKX3ASgJOABQr6Sr7AJgyf77iZikpBOgAe2c-uoDsgEOcGVybC5hYm91dC5jb23IAQHaATlodHRwOi8vcGVybC5hYm91dC5jb20vb2QvZmlsZXN5c3RlbS9hL3BlcmxfcGFyc2VfdGFicy5odG2AAgHIApP2nxCoAwG4AwHoA-AF6APdBfUDAAQAAA&num=2&sig=AOD64_02HyAIIuKRPNdssj5kCIKQuLJ38g&client=ca-primedia-premium_js&adurl=http://www.the-art-institutes.info/index.php/media_arts/audio_video_film_production/%3Feng%3Dgoogle%26kw%3Dfilm%2520school)

[**Design Your Business Site**](http://googleads.g.doubleclick.net/aclk?sa=l&ai=BA76LwYnKTpK1N86JgALvwtCkBvjLw98B0Iet2BvAjbcB4M_VARADGAMgqKX3ASgJOABQ4u708wVgyf77iZikpBOgAajAke0DsgEOcGVybC5hYm91dC5jb23IAQHaATlodHRwOi8vcGVybC5hYm91dC5jb20vb2QvZmlsZXN5c3RlbS9hL3BlcmxfcGFyc2VfdGFicy5odG2oAwG4AwHoA-AF6APdBfUDAAQAAA&num=3&sig=AOD64_0QkXWI9EuNge1vlZ3NCY_cCbbF7g&client=ca-primedia-premium_js&adurl=http://ad.doubleclick.net/clk%3B245568297%3B70547945%3Bc)Creating a website is fast and easy Free from Google to your business![www.MissouriGetOnline.com](http://googleads.g.doubleclick.net/aclk?sa=l&ai=BA76LwYnKTpK1N86JgALvwtCkBvjLw98B0Iet2BvAjbcB4M_VARADGAMgqKX3ASgJOABQ4u708wVgyf77iZikpBOgAajAke0DsgEOcGVybC5hYm91dC5jb23IAQHaATlodHRwOi8vcGVybC5hYm91dC5jb20vb2QvZmlsZXN5c3RlbS9hL3BlcmxfcGFyc2VfdGFicy5odG2oAwG4AwHoA-AF6APdBfUDAAQAAA&num=3&sig=AOD64_0QkXWI9EuNge1vlZ3NCY_cCbbF7g&client=ca-primedia-premium_js&adurl=http://ad.doubleclick.net/clk%3B245568297%3B70547945%3Bc)

**Perl Ads**

* [Open Source Perl](http://perl.about.com/z/js/o.htm?k=open%20source%20perl&d=Open%20Source%20Perl&r=http://perl.about.com/od/filesystem/a/perl_parse_tabs.htm)
* [Perl](http://perl.about.com/z/js/o.htm?k=perl&d=Perl&r=http://perl.about.com/od/filesystem/a/perl_parse_tabs.htm)
* [Script Example](http://perl.about.com/z/js/o.htm?k=script%20example&d=Script%20Example&r=http://perl.about.com/od/filesystem/a/perl_parse_tabs.htm)
* [Format Script](http://perl.about.com/z/js/o.htm?k=format%20script&d=Format%20Script&r=http://perl.about.com/od/filesystem/a/perl_parse_tabs.htm)
* [Perl Debugger Gui](http://perl.about.com/z/js/o.htm?k=perl%20debugger%20gui&d=Perl%20Debugger%20Gui&r=http://perl.about.com/od/filesystem/a/perl_parse_tabs.htm)

Let's take a minute to look at one of the reasons Perl makes a great data mining and scripting tool - parsing text files. Big or small, Perl benchmarks great when it comes to digging through text. As an example, lets build a little program that opens up a tab separated data file, and parses the columns into something we can use.

Say for example your boss hands you a file with a list of names, emails and phone numbers and wants you to read the file and do something with the information like put it into a database or just print it out in a nicely formatted report. The file's columns are separated with the TAB character and would look something like this:

Larry larry@example.com 111-1111

Curly curly@example.com 222-2222

Moe moe@example.com 333-3333

Here's the full listing we'll be working with:

#!/usr/bin/perl

open (FILE, 'data.txt');

while (<FILE>) {

chomp;

($name, $email, $phone) = split("\t");

print "Name: $name\n";

print "Email: $email\n";

print "Phone: $phone\n";

print "---------\n";

}

close (FILE);

exit;

Note that this pulls some code from the [how to read and write files in Perl](http://perl.about.com/od/perltutorials/a/readwritefiles.htm) tutorial that I've already set up. Take a look at that if you need a refresher. First it opens a file called data.txt (that should reside in the same directory as the Perl script). Then it reads the file into the catchall variable $\_ line by line. In this case, the $\_ is *implied* and not actually used in the code.

After reading in a line, any whitespace is [chomped](http://perl.about.com/od/programmingperl/qt/perl_chomp_chop.htm) off the end of it. Then the [split](http://perl.about.com/od/programmingperl/qt/perlsplit.htm) function is used to break the line on the tab character. In this case the tab is represented by the code **\t**. To the left of the split's sign, you'll see that I'm assigning a group of three different variables. These represent one for each column of the line.

Finally, each variable that has been split from the file's line is printed separately so that you can see how to access each column's data individually. The output of the script should look something like this:

Name: Larry

Email: larry@example.com

Phone: 111-1111

---------

Name: Curly

Email: curly@example.com

Phone: 222-2222

---------

Name: Moe

Email: moe@example.com

Phone: 333-3333

---------

Although in this example we're just printing out the data, it would be trivially easy to store that same information parsed from a TSV or CSV file in a full fledged database.

One more example for reading from a file writing to another and in the meanwhile doing string manipulation:

The problem is :

Hi all,  
  
Perl NEWBIE here. I created a file (that for now is filled with dummy data) called backs.dat. Basically, it just has some information about running backs in the NFL. Here's it contents:  
  
Jerome Bettis PIT 323 1460  
Priest Holmes KC 400 1500  
Kevin Faulk NE 250 1000  
Clinton Portis DEN 300 1200  
Eddie George TEN 350 1300  
Ahman Green GB 250 900  
  
Each field is seperated by tabs.  
  
I have also created a file called avgs.dat for writing. I am trying to write a script that will open the backs.dat file, reads from it, and mathematically calculates the running back averages, then sends them into the avgs.dat file. I would like any information already in the avgs.dat file to be overwritten. If either operation fails, I want the script to exit with an appropriate error message. Each line read from backs.dat should be separated so that all five fields are stored in separate variables. For each input line, a line of the following form should be output to avgs.dat:  
  
FIRST-NAME LAST-NAME AVG-YDS-PER-CARRY  
  
Note that the AVG-YDS-PER-CARRY is computed as $yards/$carries. For instance, a calculator shows me that the Jerome Bettis answer would be:  
  
Jerome Bettis 4.52012383900929  
  
After backs.dat has been processed in its entirety, both files should be closed. If a close fails, an error message should be produced and the script should exit.  
  
I am having a helluva time figuring this out. I know it's a simple calulation, but I can't seem to even get off the ground. Anyone have any thoughts on this? Could someone show me some code examples, perhaps, that would do this? There's no rush, but it is driving me crazy!  
  
Thanks!

The solution is :

#!/usr/bin/perl  
  
open (READ, "backs.dat") || die "Can't find backs.dat\n";  
  
$x=0;  
  
while ($info = <READ>) {  
chomp $info;  
@data = split (/\t/, $info);  
push @RecEnt, [@data];  
$x++;  
}  
  
close READ || die "Couldn't close backs.dat";  
open (WRITE, ">avgs.dat") || die "Can't find avgs.dat\n";  
  
$y=0;  
  
foreach (@RecEnt){  
$Avg = $RecEnt[$y][4]/$RecEnt[$y][3];  
$printout .= "$RecEnt[$y][0]\t$RecEnt[$y][1]\t$Avg\n";  
$y++;  
}  
  
print WRITE $printout;  
close WRITE || die "Couldn't close avgs.dat";  
exit (0);