

Files and Directories

chdir ("/etc")	change to directory /etc
@a = </etc/*>; @a = </etc/h*>; while (\$v= <bin/*> { @a = ~s/.*/ remove path (before last slash -- greedy)	@a gets list of files in /etc [glob] @a gets a list of h* files in /etc [glob]
opendir (ETC,"/etc") die "Cannot open dir /etc"; @a=readdir(ETC); close (ETC);	[dir handle see man readdir]
unlink ("file6"); unlink ("*.c");	remove file6 (like unix rm file6) like rm *.c (also takes lists and variables)
rename (("top","bot") die "Cannot rename top to bot"; rename ("f","bin/f");	mv, but must repeat file name in destination
link ("hat","cap"); symlink ("hat","cap"); \$x= readlink ("file");	Unix ln hat cap Unix ln -s hat cap returns name of symlink or undef (zero)

mkdir ("bin",0777) || die "cannot make dir bin" [x=1 w=2 r=4];
rmdir ("bin") || die "cannot remove dir bin";
chmod (0666,"f1","f2") Change permissions for files f1 and f2

System Processing

system ("who"); system ("who >file") && die "cannot create file right now"; return of true (nonzero) is an error -- opposite of Perl therefore && instead of	executes the shell process "who" die "cannot create file right now"; return of true (nonzero) is an error -- opposite of Perl therefore && instead of
while (system ("grep aa fl")) { push (@a, \$_) }	executes the shell process "grep" puts found lines in array @a
while (system ("grep", "aa", "fl")){same except list saves one shell push (@a, \$_) }	process; therefore faster
\$v = `grep aa fl`;	`backtics` execute the shell process "grep"
foreach (`grep aa fl`) { push (@a, \$_);}	puts found line in array @a

Regular Expressions

if (/abc/) { print "\$_"; }	search for string "abc"; print line which "abc" occurs; \$_ is the default variable
which (<>) { if (/abc/) { print "\$_"; } /ca*t/ . /c.*?t/ .*	diamond operator: this routine is like grep search for "abc" from a file or files matches "ca" any number of "a's" and "t" matches any character but \n the ? suppresses greedy: cat but not cattt any char from present to end of the line

s/cat/dogs/
s/cat/dogs/g
s/cAT/dogs/I

[Aa] [^A] [0-9] [a-zA-Z0-9] [d] [D] [w] [W] [s] [S]	match big or little A anything but A every single digit any single letter or digit digits; every digit; same as [0-9] anything not \d; same as [^0-9] words; same as [a-zA-Z0-9] same as [^a-zA-Z0-9]; any nonword char white space; same as [\r\t\n\f] sane as [^\r\t\n\f]
[a+] [a?] [a*]	one or more a's in sequence (aaaaaa) zero or one a zero or more a's in sequence

\$_ = "a bbbbb c";
s/b*/cat/;
s/b{4}/cat/;
s/b {3.7}/cat/;

s/ant(.)/bug\1/

/read|writ/ing/
\b
/cat\b/
/bcab/
/bcab\b/
/^a/
/a\$/
/a|b+/
/(a|b)+/

\1 gets paren value (\2 gets second paren)
if ants then bugs; if anto then bugo
(second parens referenced with \2)
alternative match (*reading or writing)
word boundary
"cat" but not "catalog"
"catalog" but not "concatenate"
"cat" as a word, but not in a word
matches a iff a is first char in string
matches a iff a is last char in string
match one a or any number of b's
match any number of a's or b's

\$a="real food";
\$x=\$a=~/(.)\1/;
\$a=~s/oo/ee/

\$x is 1 (true); matches oo in "food"
oo changed to ee; \$a is now "real feed";

\$1,\$2,\$3 \1\2 \3 etc can be accessed as \$1 \$2 \$3 ...

\$_ = " they cannot write at all";
/w..te/;
print \$;
print \$&;
print \$;

srand
\$n=rand(35)
\$x=@v[rand (35)]

initialize random number
Sets \$n to real number 0-34
\$x gets a random element 0-34 of @v

Perl Quick Reference

Variable and Arrays

\$var = "contents" \$v = 45 (\$a,\$b,\$c) = (2,4,6) (1..5) (\$a,\$b) = (\$b,\$a) (\$d, @list) = (\$a,\$b,\$c)	initialize a scalar variable value of \$v is 45 \$a is 2, \$b is 4, \$c is 6 same as (1,2,3,4,5) swap \$a and \$b \$d gets value of \$a, array @list gets value of \$b and \$c
@var = ("xx", "yy", "zz") \$var[0] \$var[1] \$#var	initialize an array variable recalls "xx" recalls "yy" index of last item (2 for @var)
@v = (1,2,3) @w = (0,@v,4,five) @w = (six, @w) \$b = \$w[1] \$b = ++\$w[1] \$b = \$w[1]++ @c = @w[0,1,6] @w[0,1] = (no,no) \$w[\$w] print "@w[0.\$#w]"	initialize @v (for following examples) @w is now (0,1,2,3,4,five) @w is now (six,0,1,2,3,4,five) \$b is now 0 \$b and \$w [1] are now 1 \$b is still 1 and \$w[1] is now 2 @c is now (six,2,five) @w is now (no,no,1,2,3,4,five) returns "five" (the last element) prints entire array
push (@v,\$b) pop (@v) chop (@v)	adds new element \$b to (right) end of @v removes last (rightmost) element of @v removes last char from each element
unshift (@v,\$b) shift (@v) reverse (@v) sort (@v) @v= sort{\$a<=>\$b}@v	adds new element \$b to front of @v removes first element of @v returns order of elements reversed returns elements sorted (string sort) uses a numeric sort
@v = (0,1,2,) push(@v,6) \$b = pop(@v) unshift(@v,\$b) \$b = shift(@v) @x = reverse(@v) @v = (2,3,1,11) @v = sort(@v) @v = (aa,bb,cc) chop(@v)	initialize @v (for following examples) @v is now (0,1,2,6) @v is now (0,1,2,); \$b is 6 @v is now 6,0,1,2) @v is now (0,1,2,.) \$b gets 6 again @x is (2,1,0); @v is still (0,1,2) initialize @v @v is now (1,11,2,3,) (string sort!) initialize @v @v is now (a,b,c,) [array context]
split (separator/list) \$a = "crazy-cool-cats"; @c = split (/-/,\$a); \$_ = "big blue bugs" @bugs = split	change string into array at separators; @c becomes ("crazy", "cool", "cats") \$_ and whitespace defaults
join (separator, array) \$b = join("::", @c)	change array into string with separators \$b becomes ("crazy::cool::cats"); any or no chars as separators, but no reg expressions

Hashes (Associative Arrays)

%map = ("pete", "xx", "jo", "yy", "ida", "zz")
 create associative array (pairs of values)
\$map{pete} recalls xx with key "pete" [note curly brackets]
\$map{jo} recalls yy with key "jo"
\$map {me} = "aa" creates key "me" with value "aa"
\$var{date} = 94 creates "date" with value of 94

@x = %map @x gets ("pete", "xx", "jo", "yy", "ida", "zz", "me", "aa")
%w = @x creates assoc. array from @x
keys (%map) lists keys of %map (e.g. use with *foreach*) in a scalar context returns no. of keys

each (%map) lists all current values of %map
delete \$map{jo} deletes key and value; returns the value
foreach (keys(%map)) { print (" \$map{\$_}\n"); }

String Functions

chop(\$str) discards any last char of \$str
chomp(\$str) discards \n if last char of \$str
\$v = chop(\$str) puts last char in \$v
str eq str compares two strings (true if equal)
\$var eq "this" compare contents of var with str "this"
ne, lt, gt, le, ge, cmp (returns -1, 0, or 1) these are the other string operators

\$str="ab" x 4; \$str is now "abababab"
. concatenate two strings
.= concatenation assignment strings
(\$var =~ /reg. ex./) returns true if regular expressions found
(\$var =~ /\^Pe/i) regular expression starts "pe", any case

\$var =~ s/ab/cd/; substitute -- all ab to cd (like sed)
\$var =~ tr/A-Z/a-z/; translate -- all \$var to lowercase; like Unix tr
\$count = tr/a-z//; no change but counts no. of lowercase letters
\$var = tr/a-z/ /c c complement: changes any but a-z to space
\$var = tr/a-z/ABC/d delete: deletes any match of a-z that is not abc

\$v = index(\$str,\$x) find index no of beginning string \$x in \$str
\$v = ("abc", "bc") \$v gets 1; position of "a" is 0 (zero)

\$v - rindex(\$str,\$x)) index starts from right, but numbers from left
\$v = ("cab", "c") \$v gets 3; position of first c from right

\$v = substr(\$str, \$start, \$length) \$v gets substring if found
\$start is index of string; **\$length** is no of char
\$v = substr("cab",1,3) returns "abc"; 3 (\$length) could be omitted here
\$v = substr("cab", -3,3) returns "abc"; negative counts back from right

\$str = "big boat"; initialize \$str;
substr(\$str,-4) = "payments"; \$str becomes "big payments"

Print

\$v = sprintf("%10s \n", \$str); \$v gets print string; like printf
print "hello\n" Prints "hello" to standard out
printf ("%20s %4d %6.2f\n", \$s, \$i, \$r);

Same as "C" printf; %20s for string, 4d for decimal integer, %6.2f for floating point

Control Operators

sub do_it { creates subroutine with local vars \$v and
local (\$v,@a); @a
\$v = 1} subroutine returns last expression evaluated
local(\$v,\$w) = @_; special char @_ assigns locals from parameters, elements **\$_[0], \$_[1], \$_[2]**, etc.

&do_it cats 5 *do_it* invoked with arguments (*cats* and 5)

if (expr) { if expr is true then list1 executed
statement list1
} elsif (expr2) { else if expr2 is true then list2 executed
statement list2 (can continue with more elsif)
} else { else -- when all the aboves fail execute this
statement list3
} list3

expr2 if expr; **if** statement with order reversed (same for **unless**, **while**, and **until**)

this && that; logic and; equals: if (this) {that}
this || that; logic or; equals: unless (this) {that}

if (/a/ && /e/ && /i/ && /o/ && /u/) { print "all vowels used"; }
 all conditions must be true for true; logical "and"

unless (expr) { executes *unless* expr is true
statement list } takes elsif and else (like if)

while (expr) { while expr is true repeat execution of
statement list } statement list

until (expr) { like while, but stops when expr is true
statement list }

for (ini, test, incr) { initialize a variable, test to run list,
statement list } then increment the variable

for (\$a=1; \$a<=10; \$a++) { print "\$a"; } Prints 1 through 10
for (\$a=1; \$a<=\$#var,\$a++) {print "\$a"; } 1 through length @var -1

foreach \$v (@list){ Repeats cmd list for each \$v produced
statement list by @list; **NOTE:** If you change any particular
} \$v, the element changes *in the array @list*

@w = (1..9);
foreach \$v(@w) { prints 1 through 9 on separate lines
print \$v\n;

@w = (1..9); Omits the \$v; Perl assumes the default
foreach (@w) { variable **\$_**
print \$_;

last ends loops: while, for, etc.
next skips to next item in loop -- while, for, etc.
redo jumps to top of loop; unlike *next* it doesn't get new item; use with last to break loop
LABEL7: label statements for *next* and *last* jumps for jumping out of nested loop to outer loop
last LABEL7 end nested labeled LABEL7
die "no such file"; ends program; prints message to stdout

File Operators

open (FL, "fl"); open input file fl with filehandle FL
while (<FL>){ puts next line from file fl into **\$_**
close (FL) closes file fl

open (OUT,">fl"); open file for output with filehandle OUT
open (AP,">>fl"); open file fl for append, filehandle AP

open (MAIL, " | mail fred@clarkson.edu");
 | Piping runs command -- here the mail cmd
 [put piping at end to grab cmd output |]

dbmopen (%var, "fl", 0666); keeps array %var in file fl
\$var (\$name) = time; appends time to array in fl
dbmclose(%var); 0666 sets octal file permissions

rename (\$fl, "\$fl.ex") renames *file* to *file.ex*

<STDIN> waits for keyboard input -- adds \n

<STDOUT>

<STDERR>

\$v = <STDIN> \$v gets single line input on Enter
@v = <STDIN> @v several lines; ^D to end (array context)

while (<STDIN>) { reads each line to **\$_**
print "\$_"; } **\$_** is the default variable

while (<>) { diamond operator reads @ARGV from the
print \$_; } cmd line (here it prints all lines of arg files)

File Test (list is not exhaustive)

\$fl = "filename" assigns a filename to a variable
if(-r \$fl && -w _) Underline "_" reports on a -w without
{print "use \$fl";} a new *stat* system call

-r readable (file or dir)
-w writable
-x executable
-o owned by user
-e exists
-z zero size (file exists)
-s nonzero size
-f file
-d directory
-l symlink
-T text file
-B binary file
-M modification age in days
-A access age in days
stat() remaining info on files

String Escapes for Double Quotes

\n newline
\t tab
\007 octal value (007 = bell)
\x7f hex value (7f = delete)
\\$ literal dollar sign
\l lowercase the next letter
\L lowercase letters until \E
\u uppercase next letter
\U uppercase letters until \E