Files and Directories

chdir ("*/etc*")	change to directory */etc*
@a = </etc/*>;	@a gets list of files in */etc* [glob]
@a = </etc/h*>;	@a gets a list of h* files in */etc* [glob]
while ($v= <bin/*> {	
@a = ~s/.*\///	remove path (before last slash -- greedy)

opendir (ETC,"/etc") || die "Cannot open dir /etc";
@a=readdir(ETC);

close (ETC);	[dir handle see man readdir]
unlink ("*file6*");	remove *file6* (like unix *rm file6*)
unlink ("*.c");	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");	Unix *ln hat cap*
symlink ("hat","cap");	Unix *ln -s hat cap*
$x=**readlink** ("file");	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");	executes the shell process "who"

system ("who >file") && die "cannot create file right now"; return of true (nonzero) is an error -- opposite of Perl therefore && instead of ||

while (**system** ("*grep aa fl*")) }	executes the shell process "grep"
push (@a, $_) }	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*`) {	puts found line in array @a
push (@a, $_);}	

Regular Expressions

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

s/cat/dogs/	search "cat" substitute "dogs"
s/cat/dogs/g	search every "cat" on a line, sub "dogs"
s/cAT/dogs/I	ignore case for search
[Aa]	match big or little A
[^A]	anything but A
[0-9]	every single digit
[a-zA-Z0-9]	any single letter or digit
[\d]	digits; every digit; same as [0-0]
[\D]	anything not \d; same as [^0-9]
[\w]	words; same as [a-zA-Z0-9]
[\W]	same as [^a-zA-Z0-9]; any nonword char
[\s]	white space; same as [\r\t\n\f]
[\S]	sane as [^\r\t\n\f]
[a+]	one or more a's in sequence (aaaaaa)
[a?]	zero or one a
[a*]	zero or more a's in sequence
$_ = "a bbbbb c";	
s/b*/cat/;	replaces bbbbb with cat "a cat c"
s/b{4}/cat/;	replaced 4 b's with cat: "a catb c"
s/b {3.7}/cat/;	replaces 3 to 7 b's: "a cat c" (greedy)
s/ant(.)/bug\1/	\1 gets paren value (\2 gets second paren)
	if ants then bugs; if anto then bugo
	(second parens referenced with \2)
/read|writ/ing/	alternative match (*reading or writing)
\b	word boundary
/cat\b/	"cat" but not "catalog"
/\bcat/	"catalog" but not "concatenate"
/\bcat\b/	"cat" as a word, but not in a word
/^a/	matches a iff a is first char in string
/a$/	matches a iff a is last char in string
/a|b+/	match one a or any number of b's
/(a|b)+/	match any number of a's or b's
$a ="real food";	
$x=$a=~/(.)\1/;	$x is 1 (true): matches oo in "food"
$a =~s/oo/ee/	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/;	matches "write"
print $';	$' prints "they cannot"
print $&;	$& prints "write"
print $';	$' prints "at all"
srand	initialize random number
$n=rand(35)	Sets $n to real number 0-34
$x=@v[rand (35)]	$x gets a random element 0-34 of @v

Perl Quick Reference

Variable and Arrays

$var = "contents"	initialize a scalar variable
$v = 45	value of $v is 45
($a,$b,$c) = (2,4,6)	$a is 2, $b is 4, $c is 6
(1..5)	same as (1,2,3,4,5)
($a,$b) = ($b,$a)	swap $a and $b
($d, @list) = ($a,$b,$c)	$d gets value of $a, array @list
	gets value of $b and $c
@var = ("xx", "yy", "zz")	initialize an array variable
$var[0]	recalls "xx"
$var[1]	recalls "yy"
$#var	index of last item (2 for @var)
@v = (1,2,3)	initialize @v (for following examples)
@w = (0,@v,4,five	@w is now (0,1,2,3,4,five)
@w = (six, @w)	@w is now (six,0,1,2,3,4,five)
$b = $w[1]	$b is now 0
$b = ++$w[1]	$b and $w [1] are now 1
$b = $w[1]++	$b is still 1 and $w[1] is now 2
@c = @w[0,1,6]	@c is now (six,2,five)
@w[0,1] = (no,no)	@w is now (no,no,1,2,3,4,five)
$w[$#w]	returns "five" (the last element)
print "@w[0..$#w]"	prints entire array
push(@v,$b)	adds new element $b to (right) end of @v
pop(@v)	removes last (rightmost) element of @v
chop(@v)	removes last char from each element
unshift(@v,$b)	adds new element $b to front of @v
shift(@v)	removes first element of @v
reverse(@v)	returns order of elements reversed
sort(@v)	returns elements sorted (string sort)
@v= sort{$a<=>$b}@v	uses a numeric sort
@v = (0,1,2,)	initialize @v (for following examples)
push(@v,6)	@v is now (0,1,2,6)
$b = pop(@v)	@v is now (0,1,2,); $b is 6
unshift(@v,$b)	@v is now 6,0,1,2)
$b = shift(@v)	@v is now (0,1,2,) $b gets 6 again
@x = reverse(@v)	@x is (2,1,0); @v is still (0,1,2)
@v = (2,3,1,11)	initialize @v
@v = sort(@v)	@v is now (1,11,2,3,) (string sort!)
@v = (aa,bb,cc)	initialize @v
chop(@v)	@v is now (a,b,c,) [array context]
split(/separator/list)	change string into array at separators;
$a = "crazy-cool-cats";	
@c = split (/-/,$a);	@c becomes ("crazy", "cool", "cats")
$_ = "big blue bugs"	
@bugs = split	$_ and whitespace defaults
join(separator, array)	change array into string with separators
$b = join("::", @c)	$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 = ("cabc", "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
$v = substr("cabc",1,3)	returns "abc"; 3 ($length) could be omitted here
$v = substr("cabc", -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 elsifs)
} **else** {	
statement list3	else -- when all the aboves fail execute this
}	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) { statement list }	executes *unless* expr is true takes elsif and else (like if)
while (expr) { statement list }	while expr is true repeat execution of statement list
until (expr) { statement list }	like while, but stops when expr is true
for (*ini, test, incr*) { statement list }	initialize a variable, test to run 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){ statement list }	Repeats cmd list for each $v produced by @list; **NOTE:** If you change any particular $v, the element changes *in the array @list*
@w = (1..9); foreach $v(@w) { print $v\n;}	prints 1 through 9 on separate lines
@w = (1..9); foreach (@w) { print $_;}	Omits the $v; Perl assumes the default variable $_
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>	$v gets single line input on Enter
@v = <STDIN>	@v several lines; ^D to end (array context)
while (**<STDIN>**) { print "$_"; }	reads each line to $_ $_ is the default variable
while (<>) { print $_; }	diamond operator reads @ARGV from the 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