Shell Information

command > *filename*
  write output to *filename*

command >> *file*
  append output to *filename*

command 2> *filename*
  write stderr to *filename*

command >*file* 2>&1
  write stdout and stderr to *filename*

<*filename* *command*
  input from *filename*

command$_1$ | *command*$_2$
  pipe output from *command*$_1$
  as input to *command*$_2$

command$_1$ && *command*$_2$
  execute *command*$_2$ if *command*$_1$
  has exit status zero

command$_1$ || *command*$_2$
  execute *command*$_2$ if *command*$_1$
  does not have exit status zero

$((*expression*))
  expression evaluated as arithmetic

`$0` = name of currently executing command
`$1,$2,$3,...` = command-line arguments
`$#` = count of command-line arguments
`$?` = exit status of previous command

read *varName*
  sets value of variable *varName* to
  next line read from stdin

'*str*' = *str*
"*str*" = *str* with variables interpolated
`` ` ``*command*`` ` `` = output of *command* as string

Zero exit status means true/successful
Non-zero exit status means false/failure

test *expression*
  returns *expression* result as exit status
  integer operators: -lt,-gt,-eq,-ne,-ge,-le
  string operators: =, -z, -n
  file operators: -d, -e, -f, -s, -nt

exit *Number*
  terminate script with exit status *Number*

if *Command*$_a$; then
  Commands$_1$
elif *Command*$_b$; then
  Commands$_2$
else
  Commands$_3$;
fi

case *Word* in
Pattern$_1$) *Commands*$_1$;;
Pattern$_2$) *Commands*$_2$;;
...
*)   *Commands*$_n$;;
esac

while *Command* ; do
  Commands
done

for *var* in *Word*$_1$ *Word*$_2$ ...
do
  Commands
done

```
# Display lines from file
count=0
while read line
do
   count=$((count + 1))
   echo "Line $count: $line"
done <file
```

```
# Interactively rm files in current dir
for f in *
do
   echo -n "Remove $f? "
   read answer
   if test $answer = y
   then
        echo $f
   fi
done
```

Regular Expressions

Atomic Patterns:

letters, digits, punctuation (except those below)
  match any occurrence of themselves
`\. \* \+ \? \| \^ \$ \[ \]`
  match any occurrence of the second character
`.` (dot)
  matches any single character
`(`*pattern*`)`
  matches *pattern*

Anchors:

`^`*pattern*
  matches *pattern* at the start of a line
pattern`$`
  matches *pattern* at the end of a line

Selection:

`[`*charList*`]`
  matches any single character in *charList*
`[^`*charList*`]`
  matches any single character not in *charList*
$pattern_1$`|`$pattern_2$`|`$pattern_3$`|...`
  matches any of the $pattern_i$s

*charList*s use c_1-c_2 to denote char ranges, and meta-characters lose their special meaning inside *charList*s

Repetition:

pattern`?`
  zero or one occurrences of *pattern*
pattern`*`
  zero or more occurrences of *pattern*
pattern`+`
  one or more occurrences of *pattern*

`\w` matches alphanumeric, including `'_'`
`\s` matches whitespace
`\d` matches numeric
`\b` word boundary

pattern`{`*N*`,`*M*`}`
  matches *N* to *M* occurrences of *pattern*

Perl Information

`$var`	- simple scalar variable
`$var[`*n*`]`	- n^{th} element of array
`$var{`*val*`}`	- element of hash for key *val*
`@var`	- entire array, or length in scalar context
`@var[`*i,j,k*`]`	- slice from array
`%var`	- entire hash

`'`*str*`'` = *str*
`"`*str*`"` = *str* `with variables interpolated`
`` ` ``*command*`` ` `` = `output of` *command* `as string`

  `empty string and numeric zero are FALSE`
  anything else `is TRUE`

`$_`	- default input or matched pattern
`$0`	- name of the Perl script file
`$?`	- exit status of last system command
`$$`	- process id of Perl runtime process
`@ARGV`	- command line arguments
`%ENV`	- environment variables
`%INC`	- path for included scripts

Arithmetic operators:
  `+  -  *  /  **` (power) `%` (mod) `..` (range)

Relational operators:

`==`	`!=`	`<`	`>`	`<=`	`>=`	(numeric)
`eq`	`ne`	`lt`	`gt`	`le`	`ge`	(string)
`=~`	`!~`					(pattern)

Logical operators:
  `!` (NOT) `&&` (AND) `||` (OR)
  `not` `and` `or` (low-precedence versions)

Bitwise operators:
  `~` (NOT) `&` (AND) `|` (OR) `^` (XOR)

String operations:
  `.` `concatenation`
  x `repetition`

`$`*var* `=` *expression*`;`
`$`*var*`++;` `++$`*var*`;`
`$`*var* `+=` *expr*`;` `$`*var* `-=` *expr*`; ...`
`$`*var* `=~ s/`*pattern*`/`*replacement*`/;`
`$`*var* `=~ tr/`*chars*`/`*chars*`/;`

block = { *statement*$_1$; *statement*$_2$; ... }

`while` (*condition*) *block*
`until` (*condition*) *block*
`do` *block* `while` (*condition*)
`do` *block* `until` (*condition*)
`for` (*init* ; *test* ; *next*) *block*
`foreach` $*var* (*list*) *block*

`last` - exit the loop
`next` - go to next iteration
`redo` - restart this iteration

`if` (*condition*$_1$) *block*$_1$
`elsif` (*condition*$_2$) *block*$_2$
...
`elsif` (*condition*$_n$) *block*$_n$
`else` *block*$_{n+1}$

&*subroutine*(*arglist*);
 (any of `&`, `(`, `)` can be omitted)

`sub` *name* *block*
 - subroutine definition
 - in *block*, `@_` holds args

Arithmetic:

`abs` *expr*
 returns absolute value of *expr*
`sin, cos, atan2` *expr*
 returns geometric function on *expr*
`int` *expr*
 returns integer portion of *expr*
`rand [` *expr* `]`
 returns random value in 0..*expr*
 returns random in 0..1 if no *expr*
`sqrt` *expr*
 returns square root of *expr*
`time`
 returns # seconds since Jan 1 1970

Conversions:

`chr` *expr*
 returns char represented by *expr*
`localtime` *expr*
 converts *expr* into a date/time string
`ord` *expr*
 returns ascii for first char in *expr*

Strings:

`chomp` *list*
 removes line endings from each string in *list*
`chop` *list*
 removes last char from each string in *list*
`index` *str* , *substr* [, *offset*]
 returns position of *substr* in *str* (or -1)
 and starts looking from *offset*, if given
`length` *str*
 returns # characters in *str*
`lc` *str*
`uc` *str*
 returns lower/upper case version of *str*
`lcfirst` *str*
`ucfirst` *str*
 returns *str* with 1st char in lower/upper case
`substr` *str* , *offset* [, *len*]
 returns substring of *str* starting at *offset*
 extending to end (or *len* chars, if supplied)

Arrays:

`delete` $*hash*{*key*}
 remove *key* and its value from hash
`grep` *expr* , *list*
`grep` *block* , *list*
 returns array of all elements from *list*
 for which *expr*/*block* evaluates to true
`join` *expr* , *list*
 returns a string containing all elements
 from *list* , separated by *expr*
`keys` %*hash*
`values` %*hash*
 returns an array of all keys/values in *hash*
`map` *expr* , *list*
`map` *block* , *list*
 evaluates *expr*/*block* for each element
 of list and returns array of results
`pop` @*array*
 pops off and returns last element from *array*
`push` @*array* , *list*
 pushes values of *list* onto end of *array*
`reverse` *list*
 returns the *list* in reverse order
`shift` @*array*
 pops off and returns first element from *array*
`sort` [*block* | *subr*] *list*
 returns a sorted array of values from *list*
 block/*subr* can be used to define ordering
`split` /*pattern*/ , *string*

split *string* at *pattern*s (default \s)
returns an array of split fragments

`unshift @`*`array`*`, `*`list`*

pushes values of *list* onto front of *array*

Files/Directories:

Tests (argument is either filename or filehandle)

`-r -w -x`	- file is read/write/executable
`-e -z`	- file exists, has zero size
`-s`	- file size in bytes
`-M`	- time since file modified
`-f -d`	- file is plain file, directory

`chmod` *list*

change permissions of files in *list*
first list element must be numerical mode

`link` *oldfile*,*newfile*

`symlink` *oldfile*,*newfile*

creates a link/symlink

`mkdir` *dirname*,*mode*

`rmdir` *dirname*

create/remove directory *dirname*

`unlink` *list*,

remove all files named in *list*

Input/Output:

`<`*handle*`>`

in scalar context, read next line from *handle*
in array context, read all lines from *handle*

`<>`

reads from input stream made from all files
specified in `@ARGV` or else from `STDIN`

`close` *handle*

closes the file/pipe associated with *handle*

`flock` *handle*,*op*

performs file-locking operation on *handle*
`op` is a combination of 1(shared),
2(exclusive), 4(non-block), 8(unlock)

`getc` *handle*

returns next character from *handle*

`open` *handle*,*filename*

opens a file and associates it with *handle*
conventions for specifying *filename*:

`"<`*file*`"`	open *file* for input
`"`*file*`"`	open *file* for input; == `"<`*file*`"`
`">`*file*`"`	open *file* for output and truncate
`">>`*file*`"`	open *file* for appending
`"|`*cmd*`"`	open pipe to write to *cmd*
`"`*cmd*`|"`	open pipe to read from *cmd*

`print [`*handle*`]` *expr*

displays *expr* on *handle* (`STDOUT`) stream

`printf [`*handle*`]` *fmt*,*list*

formats *list* using *fmt* and displays

System interation:

`chdir` *expr*

Changes working directory to *expr*

`die` *expr*

print value of *expr* to `STDERR` and exit

`exit` *expr*

terminate with exit status *expr*

`sleep` *expr*

suspend program execution for *expr* secs

`system` *expr*

execute *expr* as a Unix command

CGI.pm

`header()`

`return HTTP header`

`param()`

`list of parameters`

`param(`*name*`)`

`value of parameter` *name*

`param(`*name*`, `*value*`)`

`set parameter` *name* `to` *value*

`start_html, end_html`

`start_form, end_form`

`textfield, textarea, submit, hidden`

`short cuts to produce HTML`