Hard Links:

```
% ln original newLink
```

This command creates a hard link to the original file called newLink;
Both labels will refer to the same file; File will be deleted only when both labels are removed.

If newLink is a directory then links are make within the directory

Soft (symbolic) Links:

```
% ln -s original newLink
```

Symbolic/soft links can be created from one file system into another file system; Hard links are restricted to one file system

Use ls -lL to view details (which file it refers to) of the link;
ls -l will display the contents of the symbolic link;

```
- Substituting User

% su [-] userName

if userName is not specified, root is assumed.
```

```
- compress/uncompress

% compress fileName                      (.Z)
% uncompress fileName

% gzip fileName                          (.gz)
% gunzip fileName

% crypt key < sample.txt > sample.crypt    (to crypt)

% crypt key < sample.crypt > sample.txt    (to uncrypt)


key could be any string
```

- sed (stream editor) scans one or more text files and performs an editing action on all the lines that match a condition.

- actions and conditions may be stored in a file or may be specified on the command line within single quotes.

- sed commands begin with an address or an addressRange or a Regular expression.

- sed does not modify the input file; it just writes modified file to standard output

`a\`	append lines to output until one not ending in `\`
`b label`	branch to command `: label`
`c\`	change lines to following text (as in `a\`
`d`	delete lines
`i\`	insert following text before next output
`l`	list line, making all non-printing characters visible
	(tabs appear as `>`; lines broken with `\`)
`p`	print line
`q`	quit (for scripts)
`r file`	read file, copy contents to `stdout`
`s/pat1/pat2/f`	substitute `pat2` for `pat1`
	`f = g`, replace all occurrences
	`f = p`, print
	`f = w file`, write to `file`
`t label`	test: branch to `label` if substitution made to current l ine
`w file`	write line(s) to file
`y/str1/str2/`	replace each character from `str1` with corresponding
	character from `str2` (no ranges allowed
`=`	print current input line number
`!cmd`	do `sed cmd` if line is not selected
`: label`	set label for `b` and `t` commands
`{`	treat commands up to the matching `}` as a group

```
Examples:
  - substituting text:
    % sed 's/^/  /' file > file.new
    indents each line in the file by 2 spaces

    % sed 's/^ *//' file > file.new
    removes all leading spaces from each line of the file

    % sed '/a/d' file > file.new
    deletes all lines containing 'a'

    % cat sed1
    1i\
    abcd\
    efg

    % sed -f sed1 file > file.new
    will add two lines in the beginning of the file
```

```
% cat sed2
1,3c\
Lines 1-3 are censored
```

will replace lines 1-3 by Lines 1-3 are censored

```
% cat sed3
1c\
Line 1 is censored
2c\
Line 2 is censored
3c\
Line 3 is censored
```

multiple commands; individual lines are replaced

```
% sed '$r file' f > g
```

appends file at end of file f

```
% sed -e 's/^/<< /' -e 's/$/ >>/' file
```

multiple commands (-e option is optional! means script on command line)

- tr utility (translate)

```
% tr -cds string 1 string2
```

This command maps all characters in std. input from character set string1 to the corresponding character in string2; If length of string2 is less, the last character is repeated.

-c option causes string1 to be complemented (every character not in string1 now is in string1!)

-d option causes every character in string1 to be deleted from std. input

-s option causes every repeated character in string1 to be condensed to one occurence

Examples:

```
% tr a-z A-Z < file1 > file2
```

causes lower-case to upper-case conversion

```
% tr -c a X < file1 > file2
```

causes every non-a character to be replaced by X (including nl)

```
% tr -c a-z '\012' < file1 > file2
```

causes all non alphabetic characters to be replaced by ascii 12 (nl)

```
% tr -d a-c < file1 > file2
```

causes all a, b, c to be deleted from file1

Perl

```
- Shell scripts;  C programming
- Perl takes its syntax and features from both

Printing text:

  print "hello world\n";

Variables: (untyped) always begins with a $
  $i = 3;

  In addition to standard arithmetic operators, a range operator
  is available.

  print 1, 2, 3..15, "\n";          # numbers, range
  print "AB", "BBBC", "CDD", "\n";     # strings
  print "A"."B";                     # concatenation
```

```
- Arrays: dynamically allocated; array names begin with @ symbol
          array index begins at 0

  @arr = (1,2,3,4,5);
  @brr = (1..15);
  print @arr[3], "\n";
  print @arr, "\n";   # will print 12345
  print @arr + @brr;  # will print 20

- Mathematical and Logical operators:

  +, -, *, /
  ++, --
  &&, ||
```

- if-else, while, for (from C) and foreach (from C shell)

```
if ($1 == 0) {
  print "it's true\n";
}
else {
  print "it's false\n";
}

while ($i <= $j) {
  print "it's true\n";
  $i++;
}

for ($i=0; $i<10; $i++) {
  print "\$i=",$i, "\n";
}

foreach $i (1..15) {
  print "\$i=",$i, "\n";
}
```

- File I/O

```
@line = <stdin>;
foreach $i (@line);
  print "->", $i;  # also reads in EOLN; reads and prints each line
                   # from stdin
}

$FILE = "info.dat";
open(FILE);
@arr = <FILE>;  # an array of lines
close(FILE);
foreach $line (@arr) {
  print "$line";
}
```

- Functions:

```
sub pound2dollars {
  $EXCHANGE_RATE = 1.54;
  $pounds = $_[0];             # refers to the first argument
  return ($EXCHANGE_RATE * $pounds);
}


$book = 3.0;
$value = pound2dollars($book);
print "Value in dollars = $value\n";
```

- Command Line arguments

```
$n = $#ARGV+1; # number of arguments (beginning at 0)
print $n, " args: \n";
for ($i = 0; $i < n; $i++) {
  print "@ARGV[$i]\n";
}

pounds on command line:

if ($#ARGV < 0) {
  print "Specify value in pounds to convert to dollars\n";
  exit;
}
$poundvalue = @ARGV[0];
$dollarvalue = pounds2dollars($poundvalue);
print "Value in dollars = $dollarvalue\n";
```

```
 loan -a amount -p payment -r rate
 prints a table showing each month; principal and interest
  and balance

$i = 0;
while ($i < $#ARGV) {
  if (@ARGV[$i] eq "-r") {
    $RATE = @ARGV[++$i];
  }
  else {
    if (@ARGV[$i] eq "-a") {
      $AMOUNT = @ARGV[++$i];
    }
    else {
      if (@ARGV[$i] eq "-p") {
        $PAYMENT = @ARGV[++$i];
      }
      else {
        print "Unknown argument (@ARGV[$i])\n";
        exit
      }
    }
  }
  $i++;
}
```

```
if ($AMOUNT == 0 || $RATE == 0 || $PAYMENT == 0) {
  print "Specify -r rate -a amount -p payment\n";
  exit
}

print "Original balance: \$$AMOUNT\n";
print "Interest rate   : ${RATE}%\n";
print "Monthly payment : \$$PAYMENT\n";
print "\n";
print "Month\tPayment\tInterest\tPrincipal\tBalance\n\n";

$month = 1;
$rate = $RATE/12/100;
$balance = $AMOUNT;
$payment = $PAYMENT;
```

```
while ($balance > 0) {
  $interest = roundUpAmount($rate * $balance);
  $principal = roundUpAmount($payment - $interest);
  if ($balance < $principal) {
    $principal = $balance;
    $payment = $principal + $interest;
  }
  $balance = roundUpAmount($balance - $principal);
  print "$month\t\$$payment\t\$$interest\t\t\$$principal\t\t\$$balance\n";
  $month++;
}

sub roundUpAmount {
  $value = $_[0];
  $newvalue = (int (($value * 100) + 0.5))/100;
  return $newvalue;
```

```
$ perl loan.pl -r 12.5 -p 30 -a 300
Original balance: $300
Interest rate   : 12.5%
Monthly payment : $30

Month   Payment Interest        Principal       Balance

1       $30     $3.13           $26.87          $273.13
2       $30     $2.85           $27.15          $245.98
3       $30     $2.56           $27.44          $218.54
4       $30     $2.28           $27.72          $190.82
5       $30     $1.99           $28.01          $162.81
6       $30     $1.7            $28.3           $134.51
7       $30     $1.4            $28.6           $105.91
8       $30     $1.1            $28.9           $77.01
9       $30     $0.8            $29.2           $47.81
10      $30     $0.5            $29.5           $18.31
```