

```

1 Linux and Shell
Linux Commands
ls [options] [file]      List directory contents
                        -a      Include hidden files
                        -l      Long listing format
Example output:
-rwxr--r- 1 jacob sudoer 40 Jan 1 00:00 file.cpp
-          Entry type (d: directory, -: file)
rwxr--r-  Permissions (owner, group, others)
1          Number of hard links
jacob     Owner
sudoer    Group
4096      Size in bytes
Jan 1 00:00      Last modified date
file.cpp      File name
chmod [u/g/o/a][+/-/=][r/w/x] file
                        Change permissions
rm [options] file/dir   Remove file or directory
                        -r      Recursively remove
                        -f      Force remove
rmdir dir               Remove empty directory
cp [-r] src dest        Copy file or directory
                        -r      src=dir: copy dir and subtree;
                        src=dir/: copy contents of dir
                        src=file, dest=file Overwrite dest with src
                        src=file, dest=dir  Copy file into dir
mv src dest             Move file or directory
                        src=file, dest=file || src=dir, dest=dir
                        src=file, dest=dir  Renames src with dest
                        src=dir, dest=empty_dir Moves src into dest
wc [-lwcmm] file        Moves src into dest recursive
                        -l      Word count
                        -c      Count lines
                        -w      Count words
                        -m      Count bytes
                        -m      Count characters
Example output:
1 2 3 file.txt
sort [-nr|-k<n>|-t<char>] file/stdin
                        Sort lines of text
                        -n      Numeric sort
                        -r      Reverse sort
                        -k<n>    Sort by column n (1-indexed)
                        -t<char> Use char (" " by default) as delimiter
cut -d <char> [-f<n,m,...>] file/stdin
                        Get fields from lines
                        -d <char> Use char as delimiter
                        -f<n,m,...> Get fields n, m, ... (1-indexed)
uniq file/stdin         Remove adjacent duplicate lines
spell file/stdin        Get misspelt words, 1 per line
diff fileA fileB        Compare files
Example output:
0a1      Add line 1 of B after line 0 of A
> contents to be added
2,3c2    Change lines 2-3 of A to line 2 of B
< contents of A to be deleted
< contents of A to be deleted
---      separator
> contents of B to be added
4d5      Delete line 4(A) and be in sync from line 5(B)
find path [-name <name>] [-type f|d]
                        Find items under path
                        -name <name> Find items with name
                        -type f|d   Find files (f) or dirs (d) only
grep [-E <regex> | pattern] [-o] file/stdin
                        Search for pattern
                        -E <regex> Match regular expression
                        -o          Only print matching part of line
                        pattern     Match pattern
command < in_file n> out_file
                        Redirect input and output
< in_file      in_file to stdin
n> out_file     stdout(1) or stderr(2) to out_file
n>> out_file    Append to out_file
n>/dev/null     Discard stdout or stderr
2>&1           Redirect stderr to stdout
cmd1 | cmd2    Pipe stdout of cmd1 to stdin of cmd2
cmd1 && cmd2    Execute cmd2 iff cmd1 succeeds

```

```

cmd1 || cmd2    Execute cmd2 iff cmd1 fails
Shell Script
#!/bin/bash      Declares execution shell
Variables
VAR_NAME=value   No spaces around =
read VAR_NAME     Read input into variable
$VAR_NAME        Access variable
eval="$${cmd} $var" vs no_eval='${cmd} $var'
String Operations
${#a}            Length of a
${a:pos:len}     Substr from pos (0-indexed) for len
${a/find/replace} Replace first occurrence
${a//find/replace} Replace all occurrences
let "expr"       Treat values in expr as numbers
Command Line Arguments
$#              Number of arguments
$1              Access argument ($0 is script name)
${10}           Access 10th+ argument
Control Flow
if [ cond1 ]; then cmd1; elif [ cond2 ]; then
cmd2; else cmd3; fi
[ "$VAR" ]       True if $VAR is not empty
[ "$VAR" == "value" ] $VAR is equal to value
[ "$VAR" != "value" ] $VAR is not equal to value
[ "$VAR1" > "$VAR2" ] $VAR1 sorts after $VAR2
[ "$VAR1" < "$VAR2" ] $VAR1 sorts before $VAR2
[ -ef|d|s|r|w|x "$VAR" ]
-e              File $VAR exists
-f|d           File $VAR is a regular file / dir
-s            File $VAR is not empty
-r|w|x        File $VAR is read/write/executable
[ $VAR1 -eq|ne|lt|gt|le|ge $VAR2 ]
Numeric comparisons
for i in $list; do cmd; done
                $list is space-separated

```

```

2 Compiling and Makefile
CXX_COMPILER = g++      the compiler
CXX_FLAGS = -std=c++11 -pedantic-errors flags
-O0                  no optimization
-g                  add debugging symbols
-Wall               enable all warnings
-c src.cpp          compile to an object file, not linked
-o output_file      set the output file name
@                  the name of the target
$<                 the name of the first dependency
$*                 the dependencies list
$(OBJJS)            access the Makefile variable OBJJS
.PHONY: clean       make clean always run
$(CXX_COMPILER) $(CXX_FLAGS) $(OBJJS) -o main
                  compile and link the objects

```

```

3 C++ Basics
<cstdlib> (stdlib.h) and <ctime> (time.h)
srand(std::time(nullptr)); // seed generator
rand() // [0, RAND_MAX]
rand() % (b - a + 1) // [0, b - a]
rand() % (b - a + 1) + a // [a, b]
<cmath> (math.h)
double sqrt(double x);
double pow(double x, double y); // x^y
double fabs(double x); // absolute value
double ceil(double x); // round up
double floor(double x); // round down

```

```

C Arrays
T arr[n]; // array of n elements
T arr[] = {1, 2, 3}; // array of 3 elements
T arr[3][2]; // arr of 3 arrays of 2 elements
arr[0]; // access an element of an array
/* == Take array as an argument == */
ret_type func_name(T arr[], int size);
ret_type func_name(T arr[][2], int size);
/* == Linear Search == */
int linSearch(T arr[], int size, T find) {
    for (int i = 0; i < size; ++i)

```

```

    if (arr[i] == find)
        return i;
    // avg O(n) (half), worst O(n) (all)
    // add int startPos = 0; to search from startPos
    /* == Selection Sort == */
    void selSortAsc(T arr[], int size) {
        for (int i = 0; i < size - 1; ++i) {
            int minIndex = i;
            for (int j = i + 1; j < size; ++j) {
                if (arr[j] < arr[minIndex])
                    minIndex = j;
            }
            if (minIndex == i) continue;
            T temp = arr[i];
            arr[i] = arr[minIndex];
            arr[minIndex] = temp;
        }
    }
}

```

```

char and <ctype> (ctype.h)
int isdigit(int c); // nonzero if is digit
int isalpha(int c); // nonzero if is alphabet
int isalnum(int c); // nonzero if is digit or alpha
int islower(int c); // nonzero if is lower case
int isupper(int c); // nonzero if is upper case
int tolower(int c); // ret c as lower if is upper
int toupper(int c); // ret c as upper if is lower

```

```

C++ Strings and <string>
std::string str = "HKU ENGG1340 CompSc";
char c = str[5]; // 'N'
str[10] = '3'; // "HKU ENGG1330 CompSc"
str = str + " is fun"; // 0k
str = "Hello" + " World"; // Error
/* == Functions == */
// read line from istream into string until delim
std::getline(istream& in, string& s, char delim);
string::length(); // length of string
string::empty(); // true if empty
string::substr(int pos, int len);
// find first occurrence of str from pos, miss>npos
string::find(string str, int pos = 0);
// find last occurrence of str from pos, miss>npos
string::rfind(string str, int pos = npos);
// insert str before the character at pos, in place
string::insert(int pos, string str);
// erase len characters from pos, in place
string::erase(int pos, int len);
// replace len chars from pos with str, in place
string::replace(int pos, int len, string str);

```

```

File I/O and <fstream>
std::ifstream fin("input.txt");
std::ofstream fout("output.txt");
fout.open("output.txt", std::ios::app); // append
fout.open(path_str.c_str()); // only support C-strings
fout.close(); // close file
bool success = !fin.fail(); // or in.is_open();
fout << "Hi!" << std::endl; // write to file
fin >> str; // read one word from file
while (fin >> str); // read until EOF

```

```

<sstream>
std::istringstream iss("1 2 3");
iss >> a >> b >> c; // read from string

```

```

Output Formatting by <iostream> and <iomanip>
Default floating-point: 6 sig. fig., or
scientific if too large/small.
/* manipulators from <iostream> */
std::showpoint; // always show point
// # of digits depends on std::setprecision
std::noshowpoint; // unset std::showpoint
std::fixed; // fixed point notation (3.14)
std::scientific; // scientific notation (3.14e+00)
std::cout.unsetf(std::ios_base::floatfield);
// unset std::fixed and std::scientific
std::left; // left align
std::right; // right align (default with std::setw)

```

```

/* manipulators from <iomanip> */
std::setprecision(2);
// with std::fixed or std::scientific:
// 3.14159 -> 3.14 or 3.14e+00
// otherwise: 3.14159 -> 3.1
// no padding 0 without std::showpoint if
// not std::fixed or std::scientific
std::setw(10); // set width to 10
std::setfill('*'); // fill with * (char)

```

```

Pointers and Dynamic Memory
int val = 5;
int* p = &val; // p points to val
std::cout << *p; // output: 5
std::cout << p; // output: address of val
std::cout << &p; // output: address of p
p = nullptr; // p points to nothing
std::cout << *p; // SIGSEGV

```

```

int* p = new int; // allocate memory for int
delete p; // free memory
int* p = new int[10]; // allocate memory for array
delete [] p; // free memory
*p++; // increments POINTER
(*p)++; // increments VALUE

```

```

void swapByRef(int& a, int& b) {
    int temp = a; a = b; b = temp;
}
void swapByPtr(int* a, int* b) {
    int temp = *a; *a = *b; *b = temp;
}

```

```

Linked List
struct Node { int data; Node* next; };
Node* head = nullptr; // empty list
Node* n = new Node; // create a new node
n->data = 10; n->next = nullptr; // initialize
head = n; // insert first node

```

```

Node* current = head;
while (current != nullptr) { // traverse the list
    doSomething(); current = current->next;
}

```

```

void insertAtFront(Node*& head, Node* n) {
    n->next = head; head = n;
}

```

```

void insertAtEnd(Node*& head, Node* n) {
    Node* current = head;
    if (current == nullptr) {
        head = n; return;
    }
    while (current->next != nullptr)
        current = current->next;
    current->next = n; n->next = nullptr;
}

```

```

void deleteWithVal(Node*& head, int val) {
    Node* current = head, *prev = nullptr;
    while (current != nullptr) {
        if (current->data == val) {
            if (prev == nullptr)
                head = current->next;
            else
                prev->next = current->next;
            delete current;
            return;
        }
        prev = current; current = current->next;
    }
}

```

```

void reverseList(Node*& head) {
    Node *current = head, *prev = nullptr, *next;
    while (current != nullptr) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
}

```

```

void deleteList(Node*& head) {
    Node* current = head, * next;
    while (current != nullptr) {
        next = current->next;
        delete current;
        current = next;
    }
}

struct in C++
struct Point {
    int x, y, z; // member variables
    Point(int x, int y, int z) // constructor
        : x(x), y(y), z(z) {}
    double distance(const Point& p) {
        return sqrt(pow(x - p.x, 2) +
                    pow(y - p.y, 2) +
                    pow(z - p.z, 2));
    } // member function
    bool operator<(const Point& p) {
        return x < p.x
            || (x == p.x && y < p.y)
            || (x == p.x && y == p.y && z < p.z);
    } // operator overloading
};

Point p1(1, 2, 3), p2(4, 5, 6);
std::cout << "p1.X = " << p1.x << std::endl;
double dist = p1.distance(p2);

```

STL Iterators
 Iterator templates defined in <iterator>, but STL containers' iterators are defined in their own headers.

Iterator Types and Supported Operations

Forward iterators supports:

assignment, increment, dereference, equality;

Bidirectional iterators supports extra:

decrement;

Random access iterators supports extra:

addition, subtraction, inequality,
 compound (it+=1), offset dereference (it[3])

STL Containers

```

<vector>
std::vector<int> v; // declaration
/* --- Main Member Functions --- */
v.push_back(1); // append to the end, 0(1)
v.pop_back(); // remove last element, 0(1)
v[i]; // access element at i, 0(1)
v.size(); // size of vector, 0(1)
/* --- Traverse with iterators --- */
vector<T>::iterator it; // random access iterator
for (it = v.begin(); it != v.end(); ++it) {
    std::cout << *it << " ";
}

```

```

<list>
std::list<int> l; // declaration
/* --- Main Member Functions --- */
l.push_back(1); // append to the end, 0(1)
l.push_front(2); // append to the front, 0(1)
l.pop_back(); // remove last element, 0(1)
l.pop_front(); // remove first element, 0(1)
l.size(); // size of list, 0(1)
l.front(); // first element, 0(1)
l.back(); // last element, 0(1)
/* --- Traverse with iterators --- */
list<T>::iterator it; // bidirectional iterator
// same as vector

```

```

<map>
std::map<int, int> m; // declaration
m = {{key1, val1}, {key2, val2}}; // initialise
/* --- Main Member Functions --- */
m[key]; // access value with key, 0(log n)
// create if not exist
m[key] = 1; // set value with key, 0(log n)
m.count(key); // 1 if key exist, 0 otherwise
m.erase(key); // remove key, 0(log n)
m.size(); // size of map, 0(1)

```

```

/* --- Traverse with iterators --- */
map<K, V>::iterator it; // bidirectional iterator
for (it = m.begin(); it != m.end(); ++it) {
    std::cout << it->first << " ";
    std::cout << it->second << " ";
}
/* --- Use with Custom Structs S as key --- */
// S must have operator< defined

```

STL Algorithms and <algorithm>

```

std::sort(RandomAccessIterator first,
          RandomAccessIterator last);
// sort in range [first, last), in place
// O(n log n) on average
std::sort(it, it + n); // sort n elements
std::sort(v.begin(), v.end()); // sort vector
// note that the last element is not included

```

Recursion (Binary Search)

```

int binSearch(int arr[], int l, int r, int x) {
    if (r >= 1) {
        int mid = l + (r - l) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] > x)
            return binSearch(arr, l, mid - 1, x);
        return binSearch(arr, mid + 1, r, x);
    }
    return -1; // not found
}

```

4 C Language

I/O and Formatting in C (stdio. h)

```

int a; char line[100];
scanf("%d", &a); // uses format specifiers
printf("a = %d\n", a); // uses format specifiers
fgets(line, sizeof(line), stdin); // reads a line
// NULL if fail

```

Format Specifiers:

```

%d          int
%f, %lf    float, double
%.nf       float, n decimal places
%g         float, without trailing 0
%c         char
%(n)s     string (of n characters, printf() only)

```

String Functions (string. h)

```

strcpy(char[] s1, const char[] s2); // copy s2 to s1
strcat(char[] s1, const char[] s2); // s1 = s1 + s2
strcmp(const char[] s1, const char[] s2);
// -ve if s1 < s2; 0 if s1 = s2; +ve if s1 > s2
strlen(const char[] s); // length of s

```

Dynamic Memory Allocation (stdlib. h)

```

/* --- Allocate memory --- */
void* malloc(int size); // allocate size bytes
(T*) malloc(n * sizeof(T)); // = new T[n]
/* --- Release memory --- */
void free(void* ptr); // = delete ptr

```

struct and typedef

```

struct point {int x, y;};
struct point p1;
typedef struct point Point;
Point p2; // Point is alias for struct point

```

**** C-structs cannot have member functions/constructors.**

Appendix: Regular Expressions

```

^          Match the beginning of a line
$          Match the end of a line
.          Match any single character
?          Match 0 or 1 of the previous token
*          Match 0 or more of the previous token
+          Match 1 or more of the previous token
[abc]      Match any character in the brackets
[a-zA-Z]   Match any character in the range
[^abc]     Match any character not in the brackets
(p){n}     Match the pattern p for n times
(p){n,}    Match the pattern p for [n, ∞) times
(p){n,m}   Match the pattern p for [n, m] times
ab|cd      Match either ab or cd

```

```

\          Escape special characters
\d, \D     Match any digit/non-digit character
\s, \S     Match any space/non-space character
\n         Match a newline

```

```

.          Match any word/non-word character
\d, \D     Match any digit/non-digit character
\s, \S     Match any space/non-space character
\n         Match a newline

```