Chapter 1

Introduction to Linux

2013-2014 1st Semester

CSIS1123/COMP2123

Programming Technologies and Tools

Department of Computer Science, The University of Hong Kong

Slides prepared by - **Dr. Chui Chun Kit**, http://www.cs.hku.hk/~ckchui/ For other uses, please email : ckchui@cs.hku.hk

We are going to learn…

- What is Linux?
- Shell
- Some more useful shell commands
- File redirection and pipe

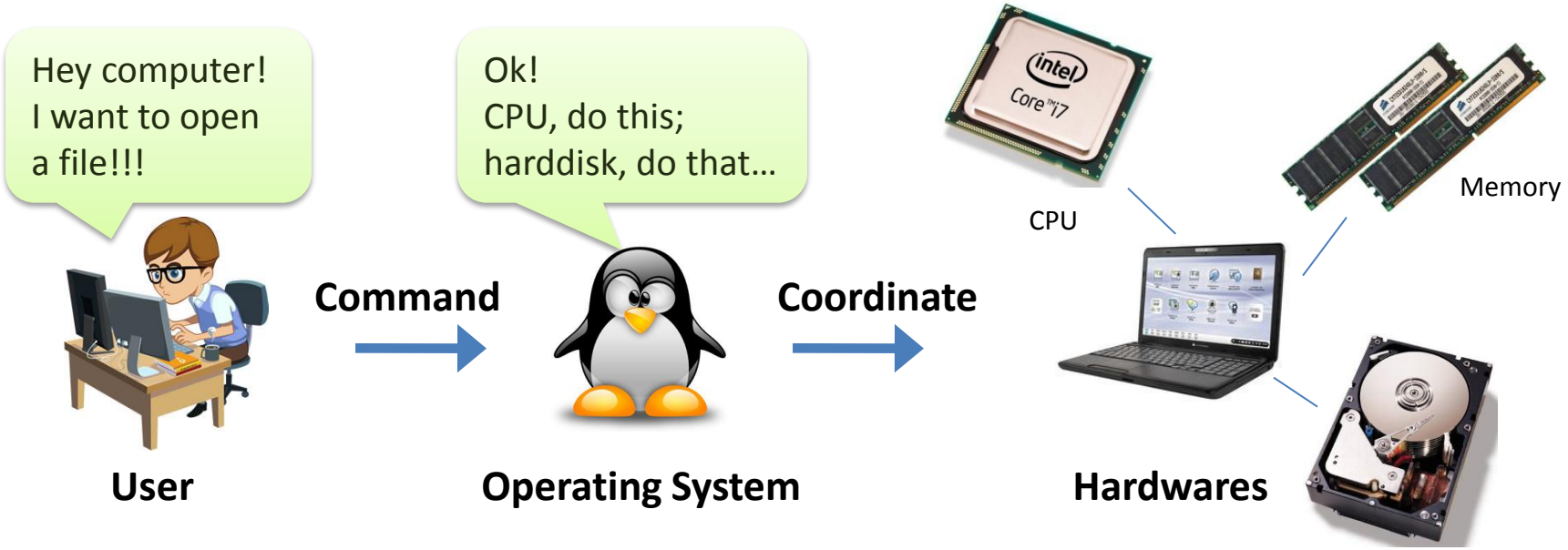
Chapter 1.1

What is Linux?

Slides prepared by - **Dr. Chui Chun Kit**, http://www.cs.hku.hk/~ckchui/ For other uses, please email : ckchui@cs.hku.hk

What is Linux?

- Linux is an **operating system (OS)** like Microsoft Windows or Apple OS X.
- OS provides an **interface** for us to use the hardware components.



What is Linux?

- There are many different distributions of Linux. They deviate from each other mainly on the interfaces and tools provided.

- This course: Fedora

Why Linux?

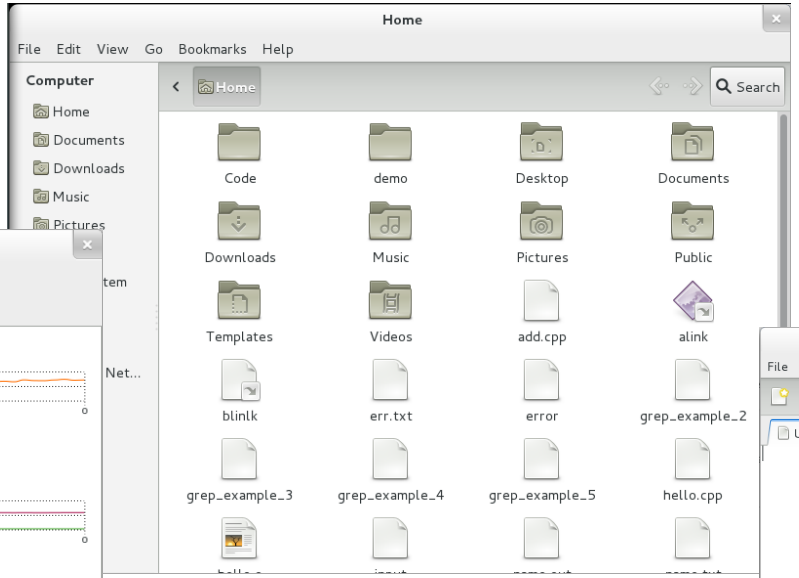
- Windows and OS X are probably more common known by the general public. However, Linux is an important operating system for a number of reasons.
  - Linux is **open-source** software. Hence, we can customize it for new or special hardware systems.

  - It provides a higher level of **security** because a company can inspect its source code.
  - Widely used for supercomputers. http://www.top500.org/lists/2012/11/

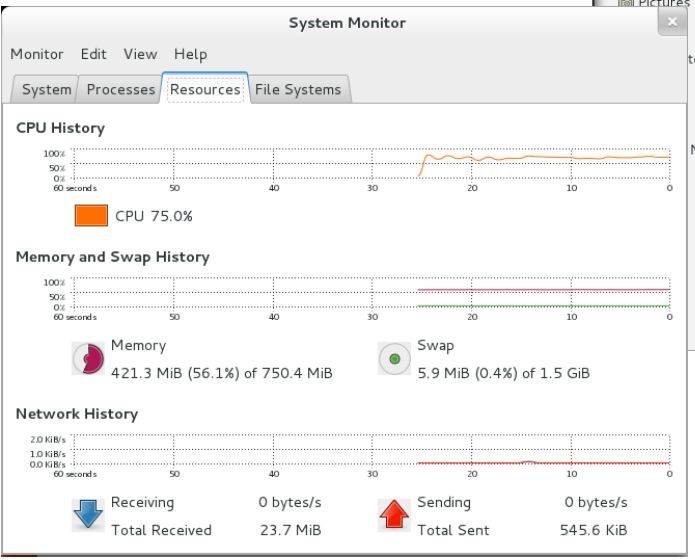
Titan (supercomputer)

Graphical User Interface

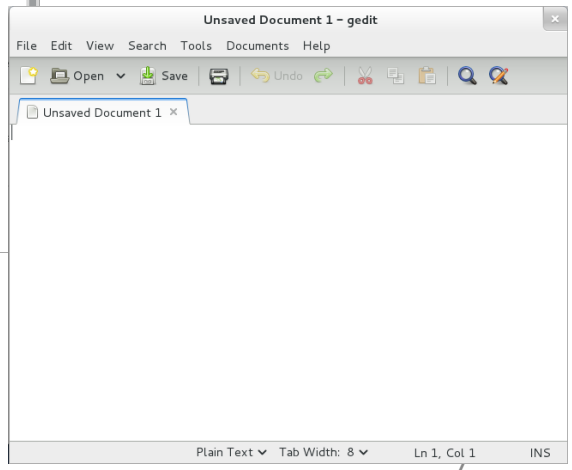
- **A GUI allows users to interact with the OS using mouse clicks, icons, menus, toolbars and other graphics elements.**



File explorer



System monitor



Text editor

Command Line Interface

- **Professional users can interact with the OS with text commands through the Command Line Interface (CLI).**
- **The program that accepts the commands is called a shell.**
- There are different shells.
  - Bash shell, Bourne Shell, Korn shell, C Shell
  - Commands are not unify among different type of shell.
  - E.g., the Korn shell uses "**print**" to print out a string, while the Bash shell uses "**echo**".

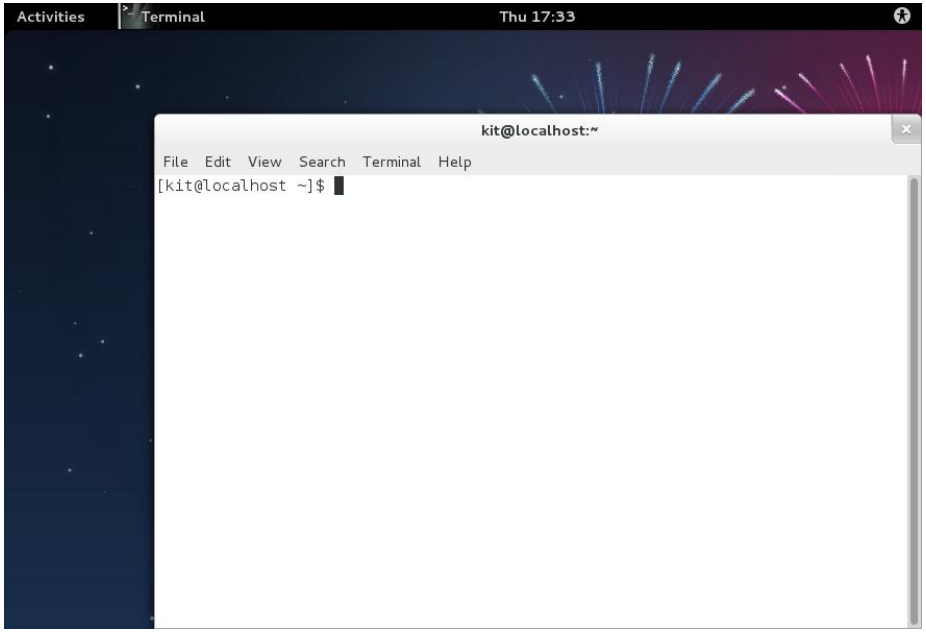
Chapter 1.2

Shell

Slides prepared by - **Dr. Chui Chun Kit**, http://www.cs.hku.hk/~ckchui/ For other uses, please email : ckchui@cs.hku.hk

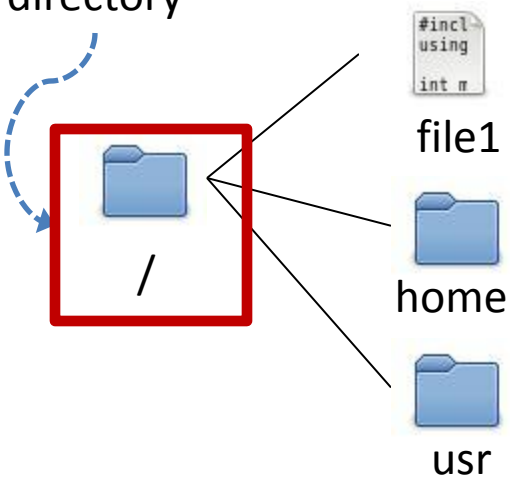
Shell

- **The default shell in Fedora is the Bash Shell**. We will focus on Bash Shell in this course.
  - You can start a shell by selecting **Applications** > **System Tools** > **Terminal**.



File system in Linux

- **Root directory** - The file structure of Linux starts with the root directory, denoted as "**/**", which contains all other files and directories.
  - To browse to the root directory, use the command **cd /**



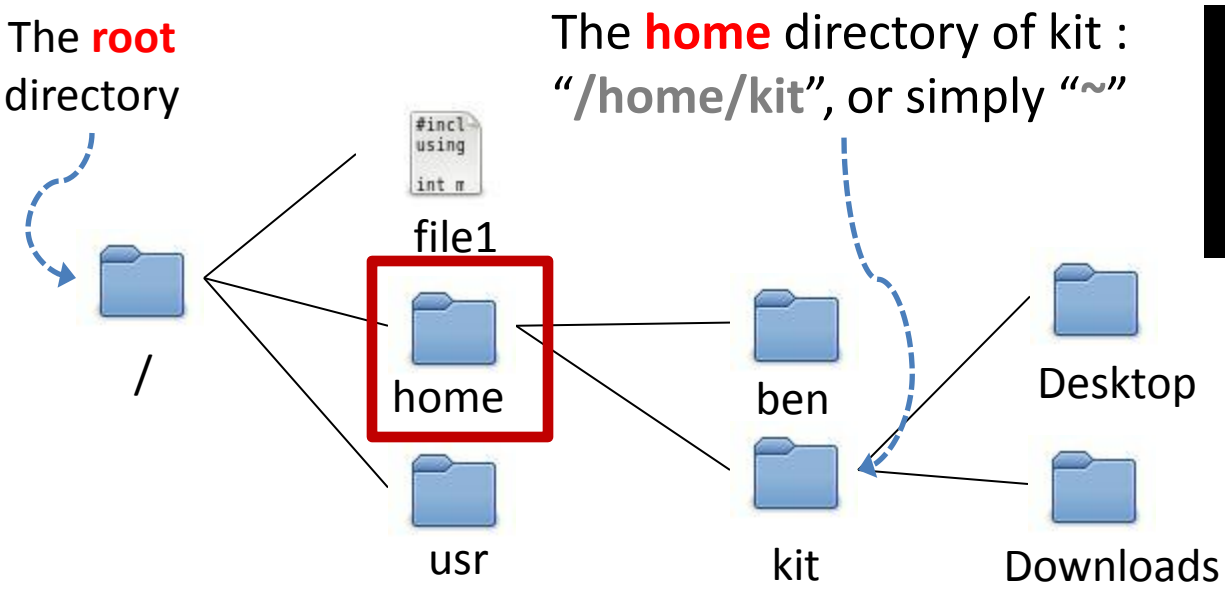

```
$cd /
$ls
file1     home     usr
```

Shell screen

The **ls** command list the directories and files in the current directory.

File system in Linux

- **Home directory** - There is a home directory for each user. For example, the user "**kit**" will have his home directory at **/home/kit**.
  - To browse to your home directory, use the command **cd ~**

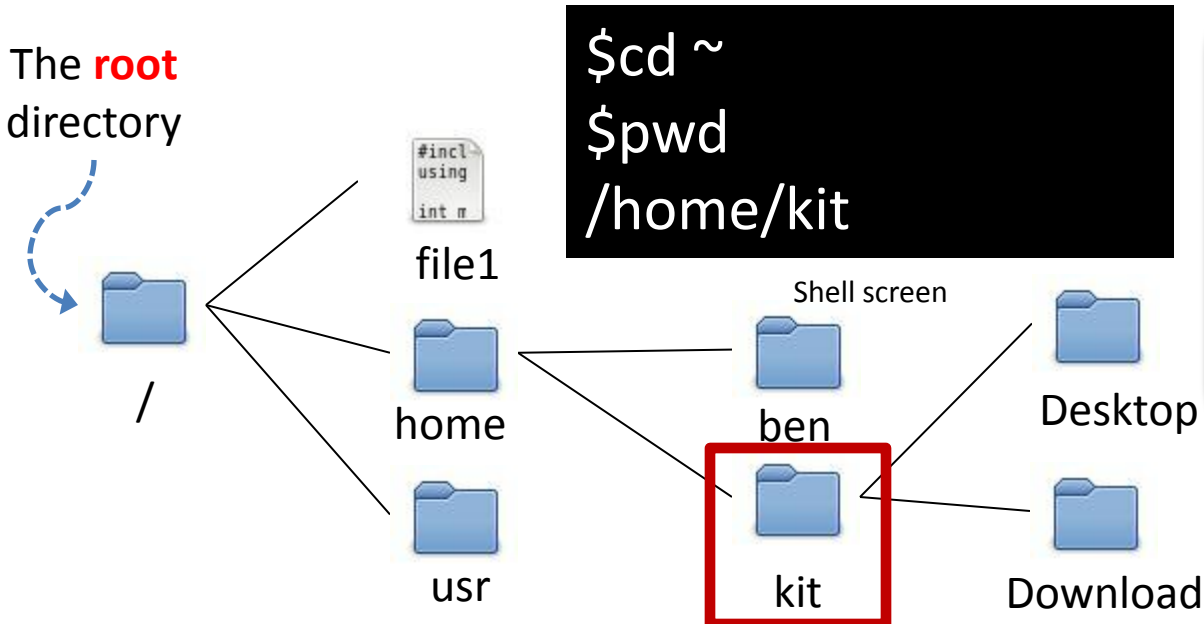



```
$cd ~
$ls
Desktop    Downloads
```

Shell screen

File system in Linux

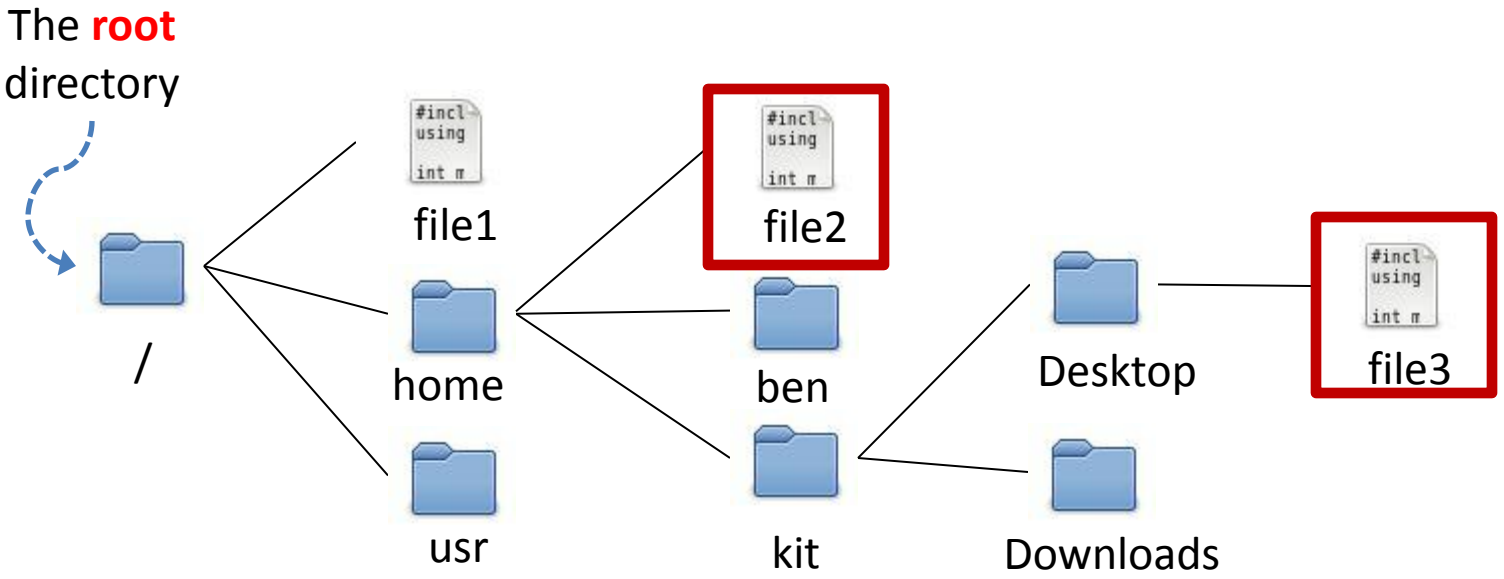
- **Present working directory** – The directory that you are currently in.
  - Once the shell starts, its present working directory is the home directory of the user.
  - To get the present working directory, use the command **pwd**.



If the user is "**kit**", then after the command **cd ~** (change directory to kit's home directory), the command **pwd** will shows the path of kit's home directory

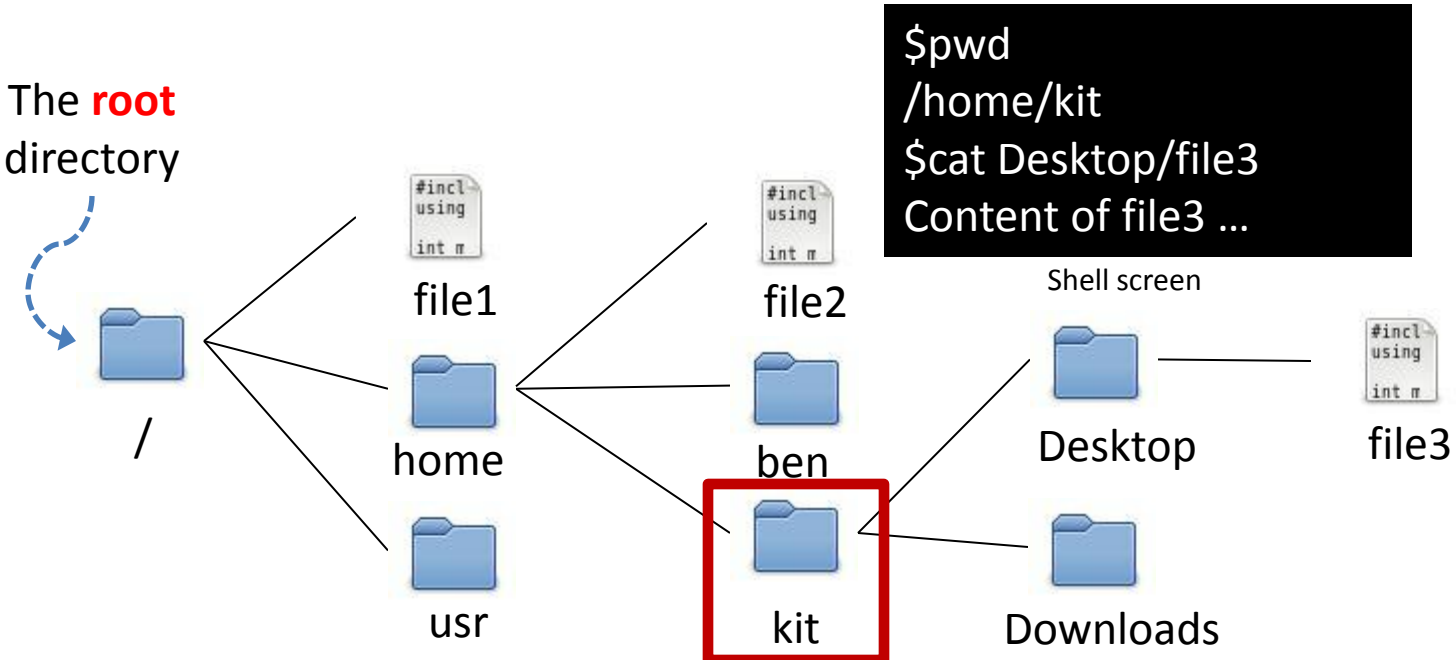
File system in Linux

- **Full path** - Full path **always start with a** "/" and concatenate all directory names from the root to the specific files.
  - **/home/file2** is the full path for file2.
  - **/home/kit/Desktop/file3** is the full path for file3.



File system in Linux

- **Relative path** – **Does not start with a "/"**. It is the path relative to the present working directory.
  - If the present working directory is **/home/kit**, then **Desktop/file3** is the relative path for file3.



Note: The command "**cat file**" displays the content of the **file**.

Directory manipulations

Command	Meaning
pwd	It prints the name of the present working directory.
ls ls -l	It lists the content in the present working directory. It lists the content in long format, which contains the file size, owners, last modification date, etc.
cd dir cd ~ cd .. cd .	It changes the current directory to dir. Changes to the home directory. Changes to the parent directory. Changes to the current directory. Hence, this command is valid yet has no effect actually.
mkdir dir	It creates a new directory with name dir.
rmdir dir	It removes the directory dir. This only works if dir is empty.
rm -rf dir	It removes the non empty directory dir and all the subdirectories & files.
mv dir dir2	If dir2 **does not exist**, it renames the directory from dir to dir2. Otherwise, it moves dir into dir2.
cp –r dir1 dir2	copy dir1 into dir2 including sub-directries

File manipulations 1

Command	Meaning
gedit a.cpp	It starts the **Gedit** program to edit the file **a.cpp**.
gedit a.cpp &	The **&** sign instructs the shell to start the program in the **background mode**. In this mode, the shell will return immediately and continue to accept other commands. Note that if there is no **&** sign, the shell will be blocked until the **gedit** program is terminated.
g++ a.cpp -o a.o	It invokes the g++ compiler to compile the program **a.cpp** into an executable **a.o**. Notice that an executable program does not need to have an extension .exe.
./a.o	It invokes the program **a.o**.

File manipulations 2

Command	Meaning
cp file1 file2	Copy file1 into file2.
mv file dir mv file1 file2 mv dir1 dir2	If dir is a directory, it moves the file into dir. If the two arguments are the same type (e.g., both file1 and file2 are files), it renames file1 to file2. The same for directories. If dir2 exists, then mv moves dir1 to dir2.
rm file rm –rf dir	Remove file. Remove recursively all files and directories in dir.
touch file	Create an empty file named file.
cat file	Display the content of file.

18

Revision

- **What is the full path of your default directory when you startup your shell?**

```
$ pwd
/student/13/cs/kit
```

Your **present working directory** should be different.

- **What are the directories in the root directory?**

```
$ cd  /
$ ls
… list of directories …
```

- **How to go back to your home directory?**

```
$cd  ~
```

Revision

- **Browse to H directory and open the Gedit editor.**

```
$ cd ~/H
$ gedit hello.cpp &
```

- **Write simple Hello World C++ code, save it, compile it.**

```
$ g++ hello.cpp -o hello.o
```

- **Run the executable.**

```
$ ./hello.o
```

Revision

- **Copy** the source code hello.cpp to hello2.cpp

```
$cp hello.cpp hello2.cpp
```

- **Rename** hello2.cpp to backup.cpp

```
$ mv hello2.cpp backup.cpp
```

- **Create** a directory "backup" and move backup.cpp in it.

```
$ mkdir backup
$ mv backup.cpp backup
```

Wildcards

- **The Linux shell has a mechanism to generate a list of file names matching a pattern**

Wildcard	Meaning
*	Matches any **string** or nothing.
?	Matches any single **character**.
[…]	Matches any one of the enclosed **characters**.

Question: What if I want to perform operation (e.g., move file) on both **hello.cpp** and **hello.o** in **ONE COMMAND**??

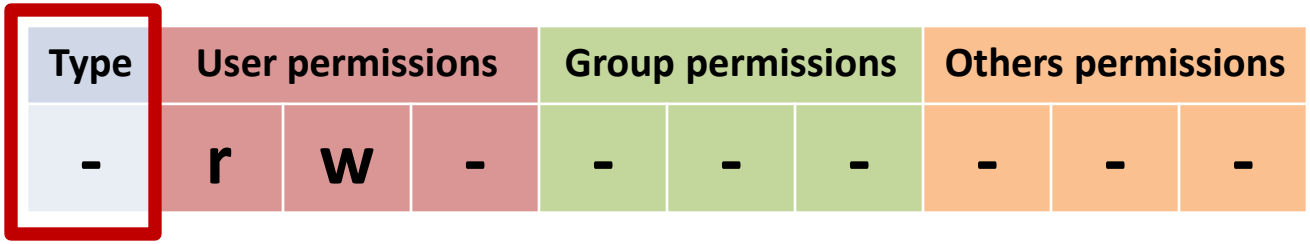
```
$mv hello.* backup
$cd backup
$ls hello.*
hello.cpp     hello.o
```

File permission & security

- **You can use the list directory command ls -l to return the permission code of files / directories.**

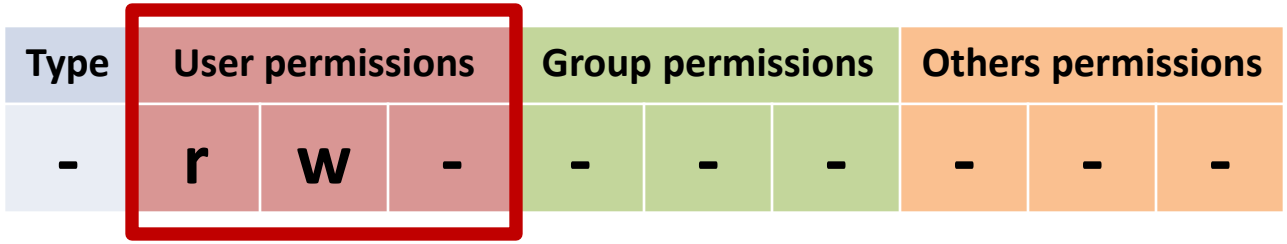
```
$ touch file
$ ls –l file
-rw-------.   1   kit   gopher   0   Jan  24 14:00   file
```

File permission & security



- **Type**
  - If it is a dash "**-**", that means it is a normal file.
  - If it is a "**d**", it means it is a directory.

File permission & security



- **User permissions**
  - 3 bits representing **Read** (**r**) , **Write** (**w**), **Execute** (**x**) permission of the file owner on the file.
  - Because the permission is "**rw-**", the owner can **R**ead and **W**rite the file, but cannot e**x**ecute the file.

File permission & security

- To change the permission of the files/ directories

chmod [who]operator[permissions] filename

who

value	meaning
u	user (owner)
g	group
o	other
a	ALL (including user, group and other)

operator

value	meaning
+	Add permission
-	Remove permission
=	Set the permission

permissions

value	meaning
r	Read permission
w	Write permission
x	Execute permission

- Grant(+) execute(x) permission to user(u):
- Grant(+) read(r) and write(w) permission to all(a):
- Remove(-) read(r) and write(w) from group(g) and other(o)

```
$ chmod u+x file
$ chmod a+rw file
$ chmod go-rw file
```

Revision

- **List the permission of the files with prefix "hello."**

```
$ ls -l hello.*
-rwx------. 1 kit gopher 76 Jan 23 9:30 hello.cpp
-rwx--x--x. 1 kit gopher 5981 Jan 23 9:30 hello.o
```

- **Take away the execute permission (x) on hello.o from user (u), what will happen?**

```
$ chmod u-x hello.o
$./hello.o
bash: ./hello.o: Permission denied
```

Chapter 1.3

More useful Shell commands

Slides prepared by - **Dr. Chui Chun Kit**, http://www.cs.hku.hk/~ckchui/ For other uses, please email : ckchui@cs.hku.hk

Summary

Command	Meaning
wc file	It **counts** the number of lines, words, and characters in **file**.
sort file	It **sorts** the lines of **file** into alphabetical order.
cut -d, -f1 file	It returns specific **columns of data**. It divides each line according to the delimiter specified by the flag **–d**, and returns the column specified by the flag **–f** (the field number starts from 1).
grep 'abc' file	It **returns the lines** in file that contains "**abc**". More sophisticated pattern matching can be specified using regular expression (Please use with the flag **-E**).
uniq file	It **removes adjacent duplicate** lines so that only one of the duplicated lines remains.
diff file1 file2	Display lines that are **different** in **file1** and **file2**. Intuitively, **diff** matches all lines that are common in both files and displays those unmatched lines.
spell file	It displays all **incorrect words** in **file**.

Word count (wc)

- The command **wc** gets the **word count information** of a **file**.

5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8

file

```
$ wc file
6 18 71 file
```

First field: 6 means there are 6 **lines** in the file.
Second field: 18 means there are 18 **words** in the file.
Third field: 71 means the file has 71 characters (bytes).

Sorting (sort)

- The command **sort** (without any other options) sorts the lines in a **file** in **alphabetical order**.

```
5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8
```

file

```
$ sort file
10 Jelly 5
1 Coke 5.5
2 Milk 8
3 Chocolate 15
4 Chicken 50
5 Apple 3.5
```

Note that the first column in the result is "**10 Jelly 5**" but not "**1 Coke 5.5**". It is because the sorting is in **alphabetical order** by default.

Numeric sort (sort -n)

- Use the flag **-n** for **numeric sort**.

5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8

file

```
$ sort -n file
1 Coke 5.5
2 Milk 8
3 Chocolate 15
4 Chicken 50
5 Apple 3.5
10 Jelly 5
```

With the flag **–n**, the product IDs (the first column) are treated as numbers in the sorting process.

Reverse order (sort -r)

- Use the flag **-r** for sorting in **reversing order**.

5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8

file

```
$ sort -n -r file
10 Jelly 5
5 Apple 3.5
4 Chicken 50
3 Chocolate 15
2 Milk 8
1 Coke 5.5
```

Question: What will be the output if we issue the command **$sort -r example**?

Answer: It will sort the **file** in **reverse alphabetical order**.

Specific sort key (sort -k)

- Use the flag **-k** for specifying **sorting key field**.

5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8

file

```
$ sort -k3 -n file
5 Apple 3.5
10 Jelly 5
1 Coke 5.5
2 Milk 8
3 Chocolate 15
4 Chicken 50
```

This command sorts the data according to the 3rd field (the price). "**-k3**" means the 3rd field as the sort key, "**-n**" means a numerical sort.

Field separator (sort -t)

- Use the flag **-t** for specifying the **field separator** if the fields are not separated by space (space is default).

5,Apple,3.5
4,Chicken,50
1,Coke,5.5
10,Jelly,5
3,Chocolate,15
2,Milk,8

file_comma

```
$ sort -t, -k3 -n file_comma
5,Apple,3.5
10,Jelly,5
1,Coke,5.5
2,Milk,8
3,Chocolate,15
4,Chicken,50
```

If the fields are separated by delimiters other than space, then we need to use "**-t,**" to specify "**,**" as the field separator.
And use "**-k**" to specify the sort key.
The above example sorts the products according to their price.

Getting column (cut)

- The command **cut** returns the specific **column** of data.
  - We need to specify the delimiter (even it is a space), the flag for specifying delimiter is **-d**.
  - We specify which column(s) to return using the flag **-f**.
  - The column IDs starts from 1 (**not 0**).

```
5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8
```

file

```
$ cut -d ' ' -f 1,3 file
5  3.5
4  50
1  5.5
10  5
3  15
2  8
```

shell

Suppose that I have a C++ program that processes the **product ID** and the **price** only…

Getting row (grep)

- **grep**, which stands for Global Regular Expression Print, is used to search in a file **and return the lines that possess the pattern.**
  - **grep** is commonly used to search for lines that contain a specific word.

Hello how are you?
I am using bash shell like a pro!

file2

```
$ grep 'pro' file2
I am using bash shell like a pro!
```

- We can specify more sophisticated patterns through **Regular Expression**.

RegEx
Regular Expression

Getting row (grep)

- Some simple **Regular Expression** syntax.

Symbol	Meaning
.	Matching any **single** character.
^	Match the **beginning** of the file only.
$	Math the **end** of the line only.
*	A single character followed by an *, will match **zero or more occurrences**.
[ ]	Character enclosed inside the [ ] will be matched. This can be a **single or range of characters**. You can use the "-" to include a range inclusively. E.g., instead of saying [12345], use [1-5].
\	Use this to **escape** the special meaning of a metacharacter. E.g., the "." Means matching any single character, we need to use "\." to mean that we are matching a dot in a pattern.
pattern { *n* }	Match ***n* occurrences** of the pattern.
pattern { *n*, }	Match **at least *n* occurrences** of the pattern.
Pattern { *n*,*m* }	Match occurrences of the pattern **between *n* and *m***.
(ab){3}	3 occurrences of the pattern '**ab**' . For example, (ab){3} will match "**ababab**", but not "**abbb**".

Getting row (grep)

- **Matching any single character.**

Hello how are you?
I am using bash shell like a pro!

file2

Conditions

Any single character	"ell"

In regular expression

.	ell

Note that **grep** returns the line that **contains** a pattern.
That's why **the whole line is returned even only "Hell" is matched by the expression**.

```
$ grep -E '.ell' file2
Hello how are you?
I am using the bash shell like a pro!
```

Use the flag **-E** when using **regular expression** with **grep**.

Getting row (grep)

apple
pineapple
apple pie

file3

```
$ grep 'apple' file3
apple
pineapple
apple pie
```

- **Matching at the beginning and ending of a line.**
  - Use '**^**' to force the pattern to match at the beginning of the line.

```
$ grep -E '^apple' file3
apple
apple pie
```

  - Use '**$**' to force the pattern to match at the end of the line.

```
$ grep -E 'apple$' file3
apple
pineapple
```

  - **^** and **$** can be used together to specify the exact content of a line.

```
$ grep -E '^apple$' file3
apple
```

Getting row (grep)

- **Matching zero or more occurrences.**
  - E.g., Matches a character '**A**' followed by zero or more occurrences of the character "**p**".

Conditions

Character A	Character p	Zero or more

In regular expression

A	p	*

Apple
Orange
Ape
Angel

file4

```
$ grep -E 'Ap*' file4
Apple
Ape
Angel
```

Getting row (grep)

- **Matching limited number of occurrences.**
  - E.g., Day is **within** one or two digits.
  - Month is **at least** 3 letters.
  - Year is **exactly** 4 digits.

2April2013
28-1-2013
13December2013

file5

Conditions

Start	Digits	Occur one or two times	Letters	Occur at least 3 times	Digits	Occur exactly 4 times

In regular expression

^	[0-9]	{1,2}	[a-zA-Z]	{3,}	[0-9]	{4}

```
$ grep -E '^[0-9]{1,2}[a-zA-Z]{3,}[0-9]{4}' file5
2April2013
13December2013
```

Getting row (grep)

```
2011111111,John,M,98
2011222012,Marry,F,85
2011333333,Sally,F,85
2012111111,Kit,M,86
2012222222,Ben,M,97
2012333333,Smitty,F,92
2012444444,Jolly,F,93
2012555555,Ken,M,100
```

data

```
$ grep 'Kit' data
2012111111,Kit,M,86
```

The line containing Kit's record should have the **substring** "**Kit**", so the Regular Expression is simply "**Kit**".

Find **Kit**'s record for me please!

Getting row (grep)

```
2011111111,John,M,98
2011222012,Marry,F,85
2011333333,Sally,F,85
2012111111,Kit,M,86
2012222222,Ben,M,97
2012333333,Smitty,F,92
2012444444,Jolly,F,93
2012555555,Ken,M,100
```

data

```
$ grep -E '^2012' data
2012111111,Kit,M,86
2012222222,Ben,M,97
2012333333,Smitty,F,92
2012444444,Jolly,F,93
2012555555,Ken,M,100
```

Find the records of the students in year **2012** (i.e. The students with "**2012**" as the **prefix** of their **UID**)

Condition

Start	2012

In regular expression

^	2012

Getting row (grep)

Condition

Start	2012	Any characters	,	Either S or J

In regular expression

^	2012	.*	,	[SJ]

```
2011111111,John,M,98
2011222012,Marry,F,85
2011333333,Sally,F,85
2012111111,Kit,M,86
2012222222,Ben,M,97
2012333333,Smitty,F,92
2012444444,Jolly,F,93
2012555555,Ken,M,100
```

data

```
$ grep -E '^2012.*,[SJ] ' data
2012333333,Smitty,F,92
2012444444,Jolly,F,93
```

Find the students who are from **2012** and with name start with “**S**” or “**J**”.

Remove duplicate (uniq)

- The **uniq** command **removes adjacent duplicate lines** so that only one of the duplicated lines remains.

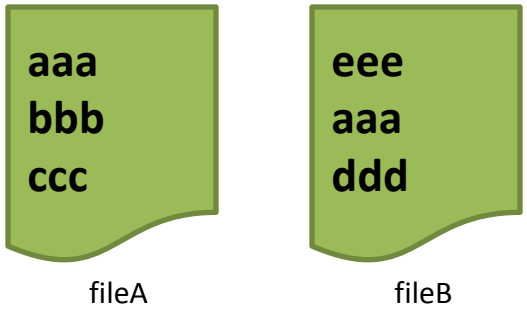
Apple
Apple pie
Apple pie
Apple
Apple
Apple pie

file6

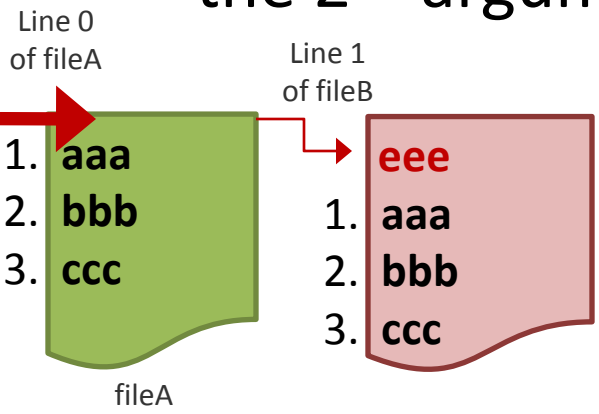
```
$ uniq file6
Apple
Apple pie
Apple
Apple pie
```

Note that **uniq** just removes **adjacent duplicates.**

Difference (diff)



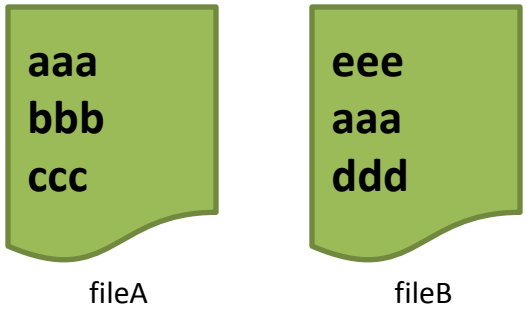
- The output of **diff** is actually a description of how to transform the file in the 1st argument into the file to the 2nd argument.



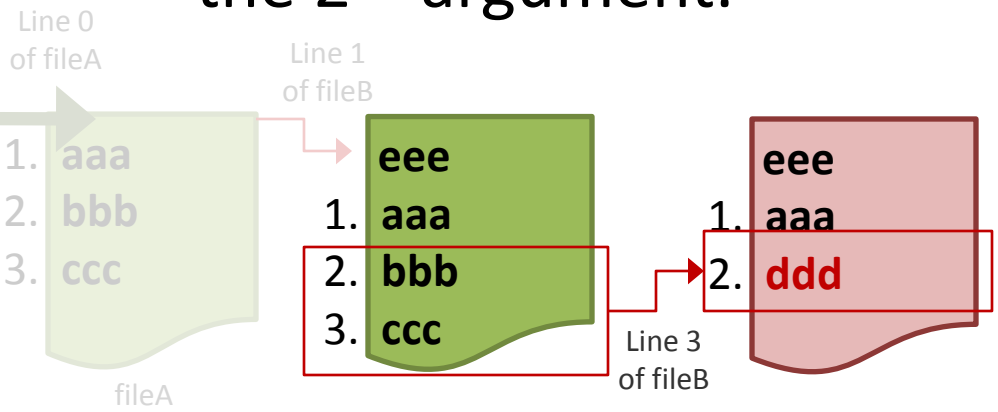

```
$diff  fileA  fileB
0a1
> eee
```

0a1– To **add** a line at line **0** of **fileA**, the line to add is line **1** of **fileB**.

Difference (diff)



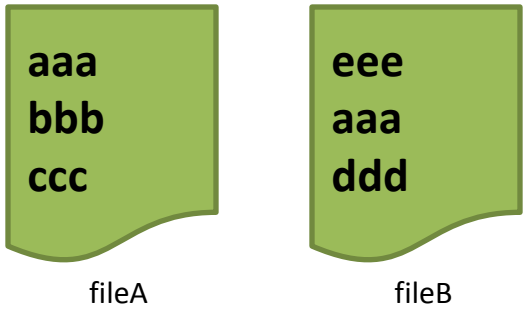
- The output of **diff** is actually a description of how to transform the file in the 1st argument into the file to the 2nd argument.



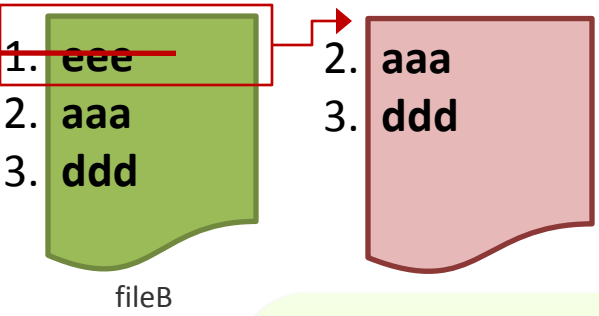

```
$diff  fileA  fileB
0a1
> eee
2,3c3
< bbb
< ccc
---
> ddd
```

2,3c3 – To **change** line **2,3** of **fileA**, to line **3** of **fileB**.

Difference (diff)



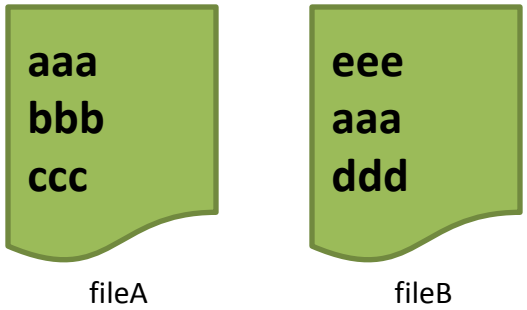
- The output of **diff** is actually a description of how to transform the file in the 1st argument into the file to the 2nd argument.



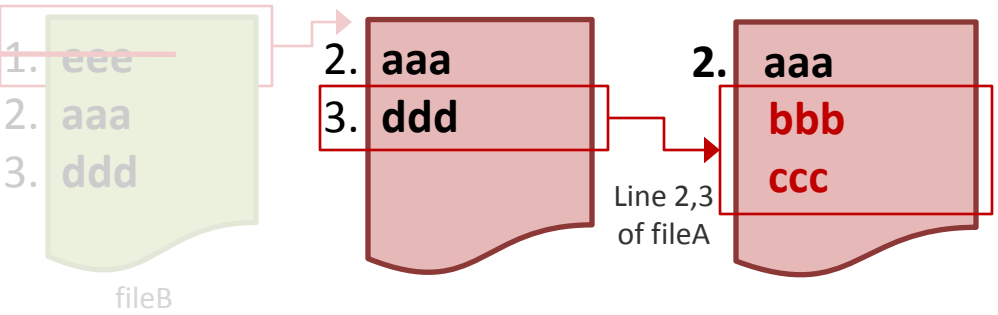

```
$diff  fileB  fileA
1d0
< eee
```

1d0 – To **delete** line **1** from **fileB**, and the files will then be in sync, starting at line **0**.

Difference (diff)



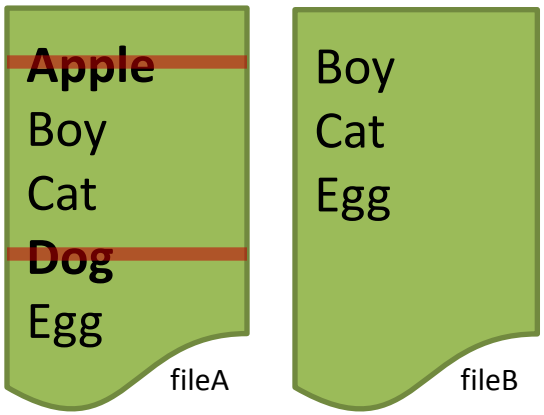
- The output of **diff** is actually a description of how to transform the file in the 1st argument into the file to the 2nd argument.




```
$diff  fileB  fileA
1d0
< eee
3c2,3
< ddd
---
> bbb
> ccc
```

3c2,3 – To **change** line **3** of **fileB** to line **2,3** of **fileA**.

Difference (diff)




```
$diff fileA fileB
1d0
< Apple
4d2
< Dog
```

```
$diff fileB fileA
0a1
> Apple
2a4
> Dog
```

Doubt:
I understand how **diff** works, but **why** the output is claimed to be the difference between two files?

Think in this way:
If we remove lines from **fileA** to create **fileB**, then **diff** will output lines that are not common among the two files (i.e., their differences).

Checking spelling (spell)

- The **spell** command displays all incorrect words in **file**.

It's a beautiffful day!
I am so happpy todday.

file7

```
$ spell file7
beautiffful
happpy
todday
```

If your Fedora does not have the command spell, please install it by:
$ **su** (switch to **root** account)
$ **yum install aspell** (install the **aspell** package)
$ **exit** (to go back to your original user account)

Chapter 1.4

File redirection and PIPE

Slides prepared by - **Dr. Chui Chun Kit**, http://www.cs.hku.hk/~ckchui/ For other uses, please email : ckchui@cs.hku.hk

File redirection >

- When using commands, the output is printed to the screen.
- We can **redirect the output to a file** using the file redirection operator **>**.

```
$ls –l
total 240
-rwx------. Kit  gopher 242 Jan 28  16:00 hello.o
-rw-------. Kit  gopher 444 Jan 28  15:30 hello.cpp
```

No file redirection

I want to store the directory information in a file, can I do it with the command **ls -l**?

```
$ ls -l  > result.txt
$ cat result.txt
total 240
-rwx------. Kit  gopher 242 Jan 28  16:00 hello.o
-rw-------. Kit  gopher 444 Jan 28  15:30 hello.cpp
```

The output of a command is directed to result.txt

```
total 240
-rwx------. Kit  gopher 242 Jan 28  16:00 hello.o
-rw-------. Kit  gopher 444 Jan 28  15:30 hello.cpp
```

result.txt

File redirection <

- Similarly, if we have a program or command that accepts user inputs, we can **redirect the file as input to the program or command** by the redirection operator **<**.

```
//add.cpp
#include <iostream>
using namespace std;
int main(){
  int a, b, c;
  cin >> a >> b >> c;
  cout << a + b + c << endl;
}
```

add.cpp

```
3 4 5
```

input.txt

```
$ ./add.o < input.txt
12
```

Redirect input.txt as input to add.o

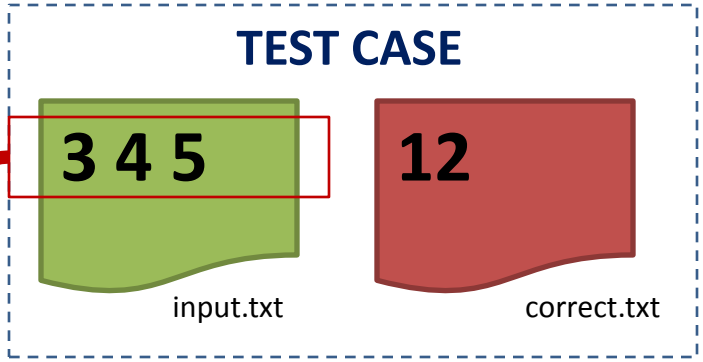
Note: it is the **cin** gets the values from the file **input.txt**. **Do not** mix this technique with C++ file I/O (**fstream** objects).

Example

- This technique is very useful to test your program against many test cases!

```
//add.cpp
#include <iostream>
using namespace std;
int main(){
  int a, b, c;
  cin >> a >> b >> c;
  cout << a + b + c << endl;
}
```

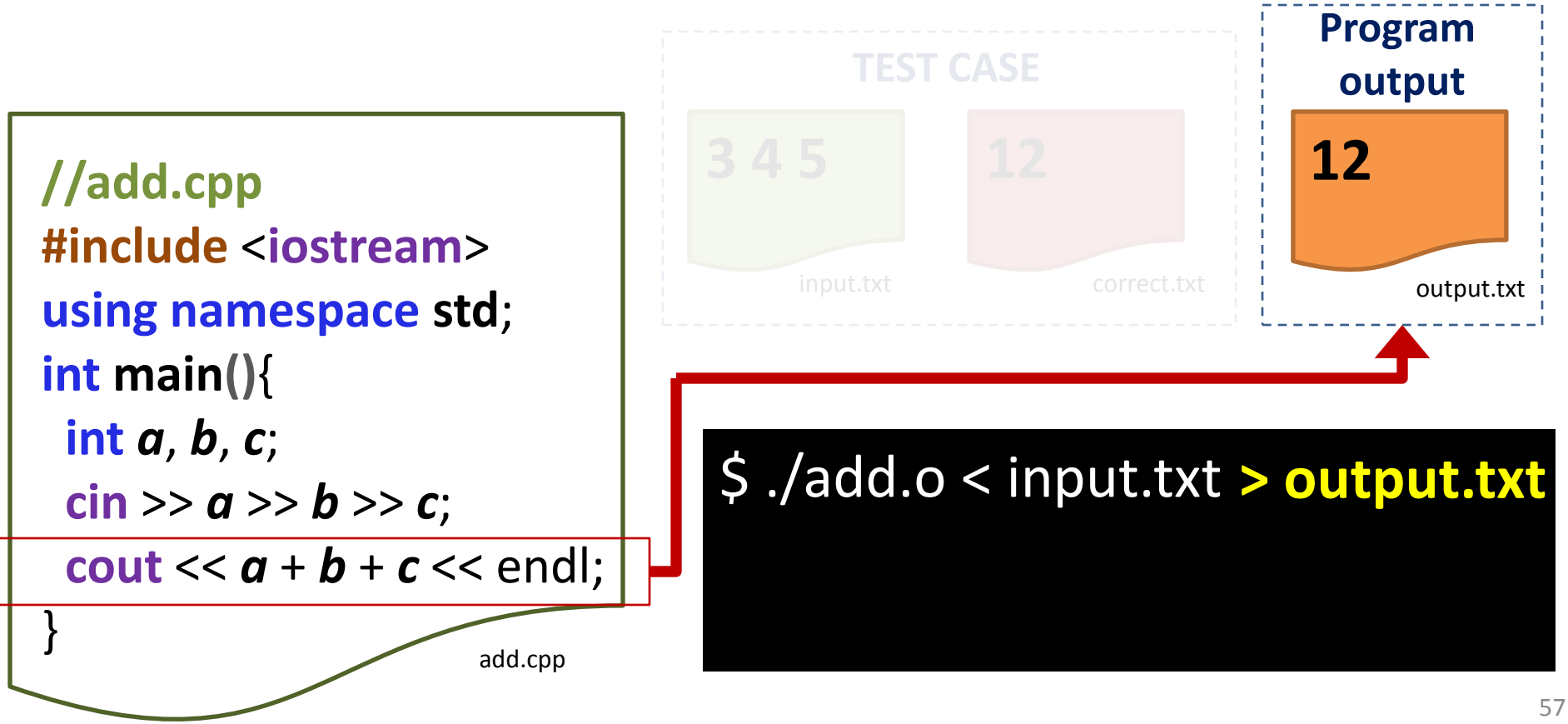
add.cpp




```
$ ./add.o < input.txt
```

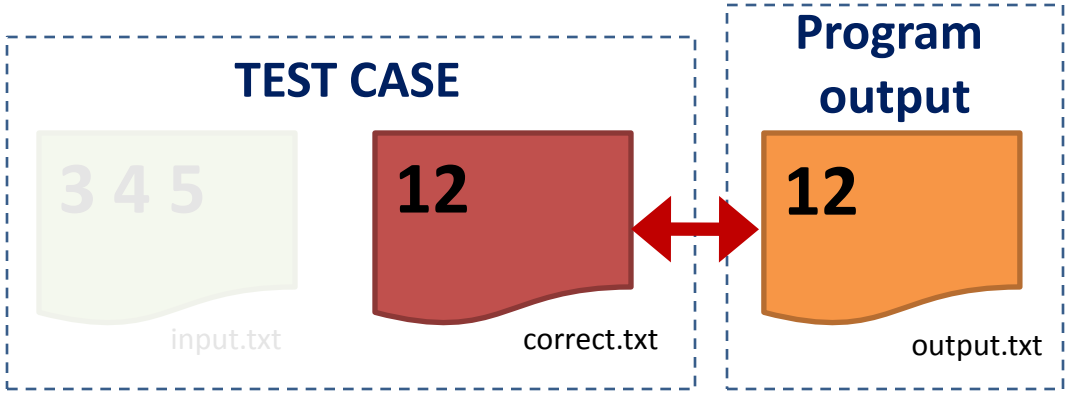
Example

- Redirects the test case **input.txt** to the program, and redirects the program output to **output.txt**.



Example

- If **diff** between the sample output and the program output returns nothing, then the program passes the test case.




```
//add.cpp
#include <iostream>
using namespace std;
int main(){
  int a, b, c;
  cin >> a >> b >> c;
  cout << a + b + c << endl;
}
```

add.cpp

```
$ ./add.o < input.txt > output.txt
$ diff output.txt correct.txt
$ (Nothing)
```

Summary

Command	Explanations
command < file	command gets its input from file.
command 1> file or command > file	Redirects command's **standard output** to file.
command 1>> file or command >> file	Redirects command's **standard output** to file (Append).
command 2> file	Redirects command's **standard error** to file.
command 2>> file	Redirects command's **standard error** to file (Append).
command < fileA > fileB 2>fileC	command gets its input from fileA and sends output to fileB, error to fileC.

59

PIPE

- Sometimes, we want to redirect the output of a program as the input of another program.
  - E.g., To find all files created in 4pm, we can use a temporary file **file.txt** to store the result of the **ls** command.
  - Then, we can **grep** those lines containing the pattern "**16:00**".

```
$ls -l > files.txt
$grep "16:00" < files.txt
```

But this technique creates an intermediate file **files.txt**.
Is there a way to streamline a sequence of commands?

PIPE

- There is a more convenient way by using a **pipe** (i.e., the "|" symbol), **which redirects the output of one program directly into the input of another program** (No intermediate files needed).

```
$ls -l | grep "16:00"
```

ls -l

```
240
-rwx------. Kit  gopher 242 Jan 28  16:00 hello.o
-rw-------. Kit  gopher 444 Jan 28  15:30 hello.cpp
dr--------. Kit  gopher 35 Jan 28  13:00 backup
-rw--------. Kit gopher 444 Jan 24  16:00 test.txt
dr--------. Kit  gopher 35 Jan 28  11:00 Desktop
```

All files

Serves as input of

grep "16:00"

```
-rwx------. Kit  gopher 242 Jan 28  16:00 hello.o
-rw--------. Kit gopher 444 Jan 24  16:00 test.txt
```

All files created in 4pm

Example 1

- To sort the products by their price and stores only the **product name** and **product price** in the file **result.txt**:

```
$ sort -k3 -n file
```

5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8

file

SORT

5 Apple 3.5
10 Jelly 5
1 Coke 5.5
2 Milk 8
3 Chocolate 15
4 Chicken 50

The output of the command **sort**

Example 1

- To sort the products by their price and stores only the **product name** and **product price** in the file **result.txt**:

```
$ sort -k3 -n file | cut -d' ' -f2,3 > result.txt
```

5 Apple 3.5
4 Chicken 50
1 Coke 5.5
10 Jelly 5
3 Chocolate 15
2 Milk 8

file

SORT

5 Apple 3.5
10 Jelly 5
1 Coke 5.5
2 Milk 8
3 Chocolate 15
4 Chicken 50

The output of the command **sort**

CUT

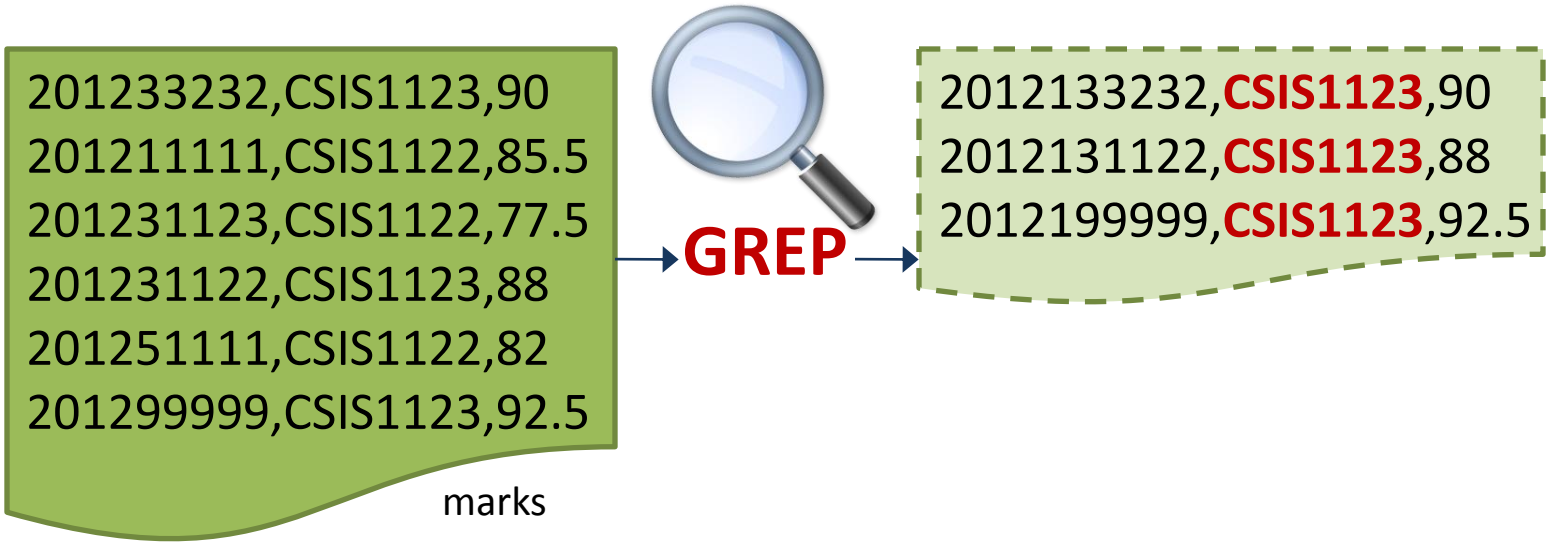
Apple 3.5
Jelly 5
Coke 5.5
Milk 8
Chocolate 15
Chicken 50

The result of **cut** redirected to **result.txt**

Example 2

- To get the marks of students in **CSIS1123** and returns their average.

```
$ grep "CSIS1123" marks
```



Example 2

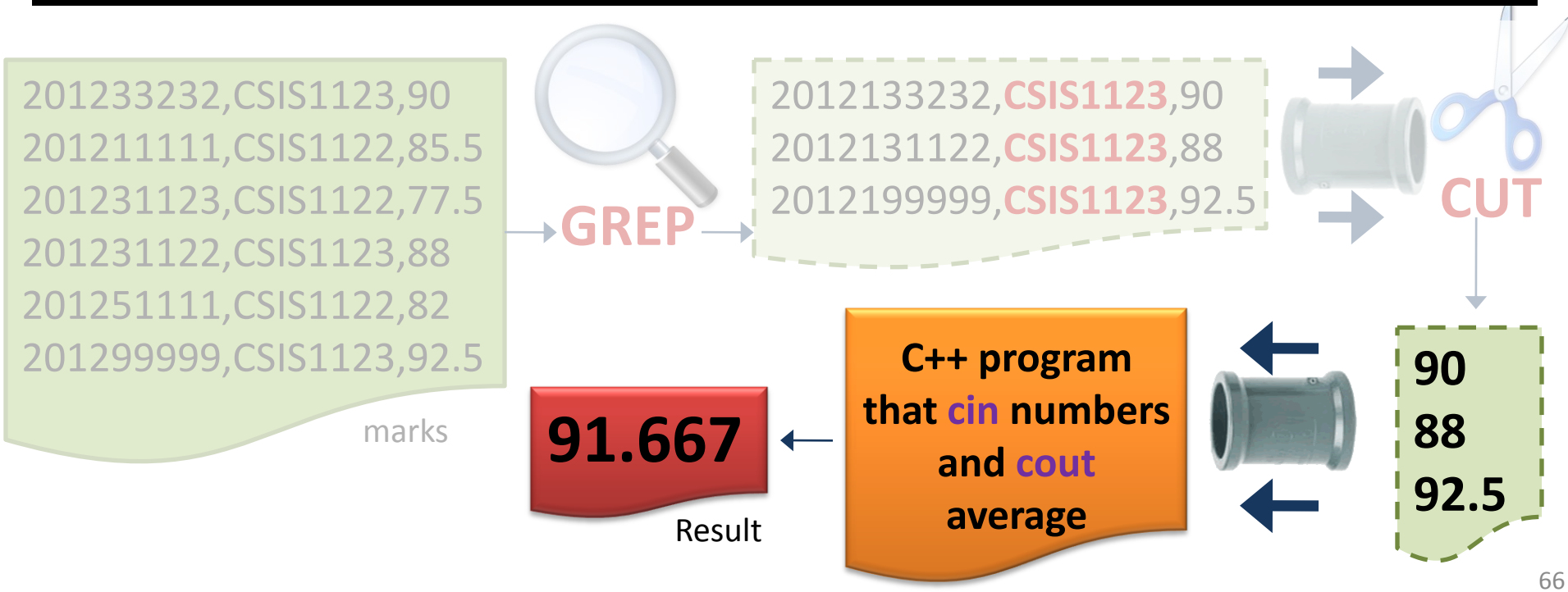
- To get the marks of students in **CSIS1123** and returns their average.

```
$ grep "CSIS1123" marks | cut -d',' -f3
```

Example 2

- To get the marks of students in **CSIS1123** and returns their average.

```
$ grep "CSIS1123" marks | cut -d',' -f3 | ./average.o > result
```



We are happy to help you!

- Please let us know if you cannot follow the self-learning materials.
- If you face any problems in this course, please feel free to contact me (Kit) or our TAs we are very very happy to help you ☺.

Chapter 1

END

2013-2014 1st Semester
CSIS1123/COMP2123
Programming Technologies and Tools

Slides prepared by - **Dr. Chui Chun Kit**, http://www.cs.hku.hk/~ckchui/ For other uses, please email : ckchui@cs.hku.hk