

# Introduction of Linux

Huang Cheng-Chao  
Dept. of Comput. Sci. & Tech.  
East China Normal University

## PART I

- Brief Introduction
- Basic Conceptions & Environment
- Basic Commands
- Shell Script

## PART II

- Text Editor (Vim)
- Compile & Debug (for C)

## PART III

- Install & Configure a Virtual Machine

## PART I

- **Brief Introduction**
- Basic Conceptions & Environment
- Basic Commands
- Shell Script

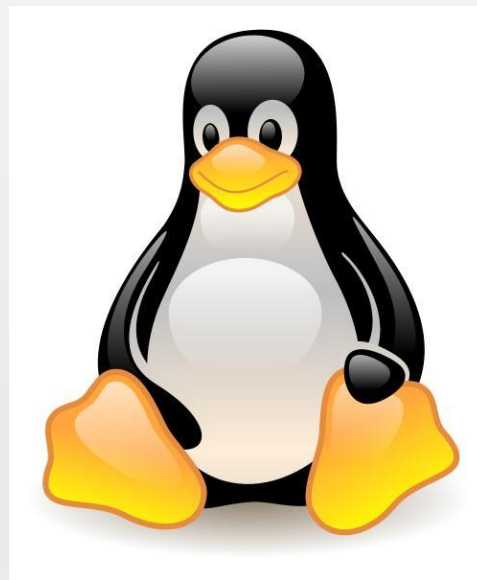
# Brief Introduction

Linux ( /'linəks/ )

a **open-source** Unix-like computer operating system originally created by **Linus Torvalds** with the assistance of developers around the world.



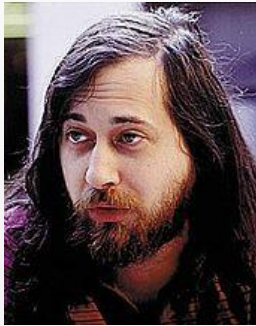
Linus Torvalds



Torvalds Unix

# Brief Introduction

## History



1983  
GNU Project  
(GNU's Not Unix)  
Richard Stallman

1991  
Linux Kernel  
Linus Torvalds

1969  
UNIX OS  
AT&T Bell Laboratory  
Ken Thompson,  
Dennis Ritchie

1970s  
BSD (Berkeley Software Distribution)

1987  
MINIX OS (for education)  
Andrew S. Tanenbaum



Linux Distributions  
· RedHat  
· Fedora  
· Suse  
· Debian  
.....

# Brief Introduction

## Widely Used

Be widely used in business, education or scientific research.

96.55% of web servers run Linux (May 2015)



for Mobile Devices



for Big Data & Cloud Computing

## PART I

- Brief Introduction
- **Basic Conceptions & Environment**
- Basic Commands
- Shell Script

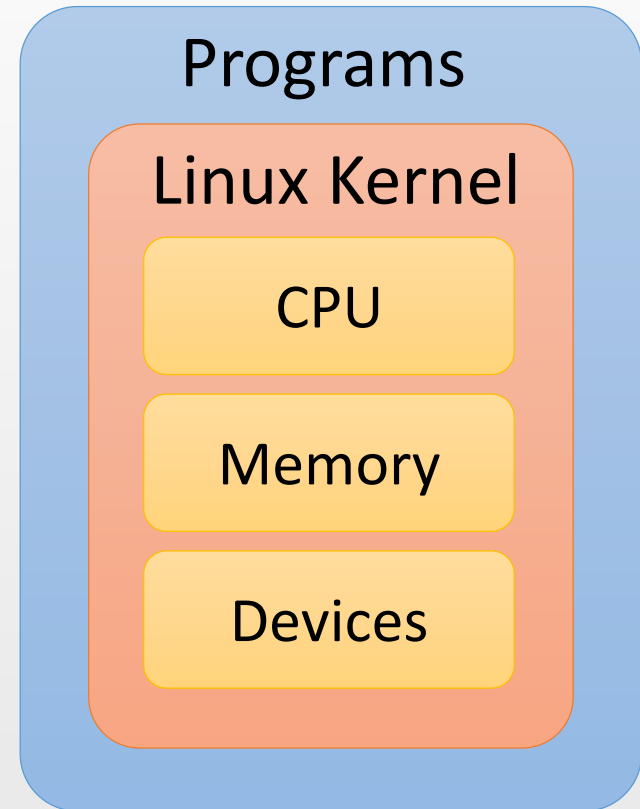
# Basic Conceptions & Environment

## Linux Kernel

The most important component of Linux OS, containing all the operating system's **core functions** and the **device drivers**

- memory management
- process scheduling
- file system

.....





# Basic Conceptions & Environment

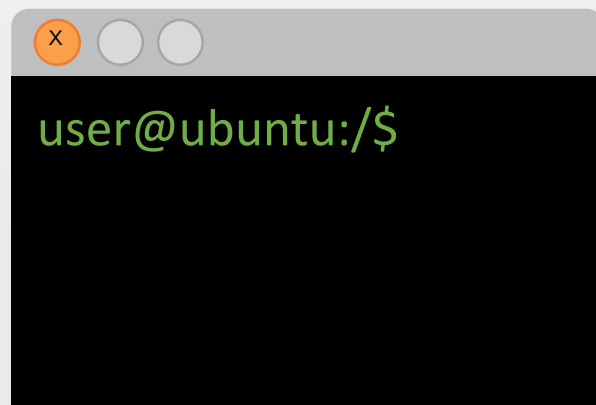
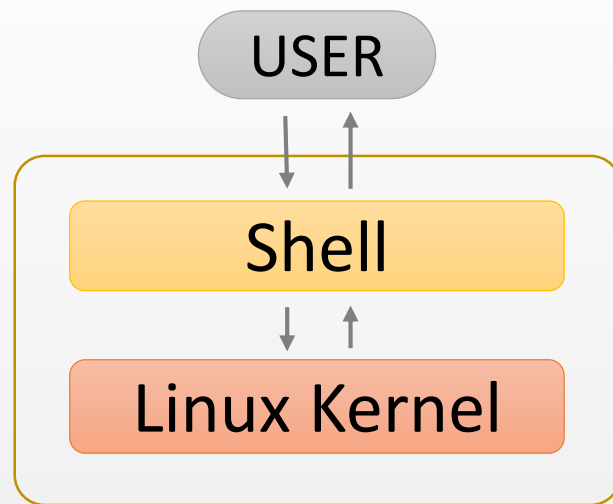
Shell (CLI shell)

Command Line Interface

A **program** which accepts commands as text input and **converts commands** to appropriate operating system functions.

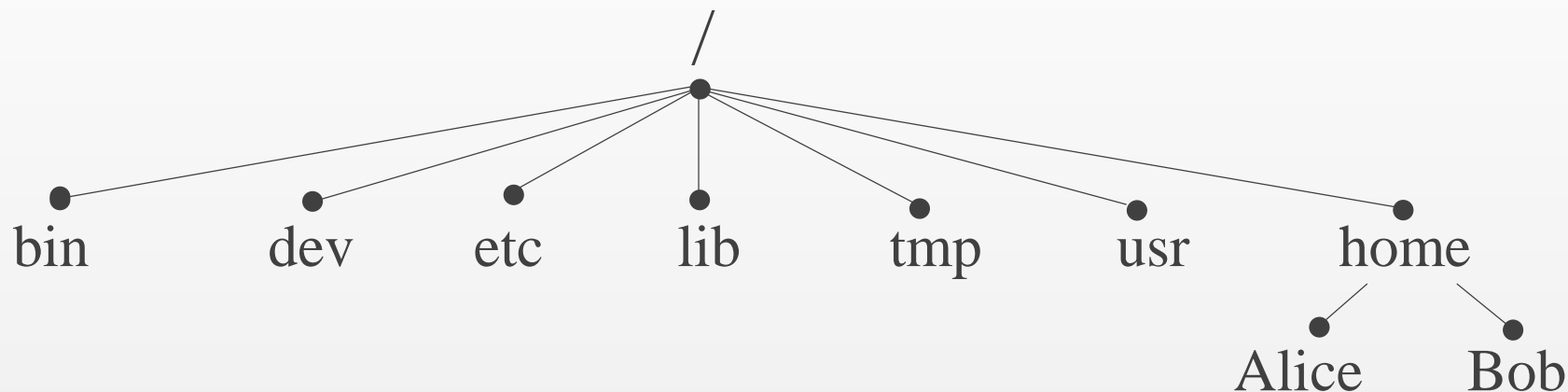
Terminal  $\leftrightarrow$  Shell

The terminal send information to the shell, receive and display the information from the shell



# Basic Conceptions & Environment

## File System



tree structure, with the **root directory** “ / ”

each node is either a file or a directory of files

full path name            /home/Alice/..... (start from / )

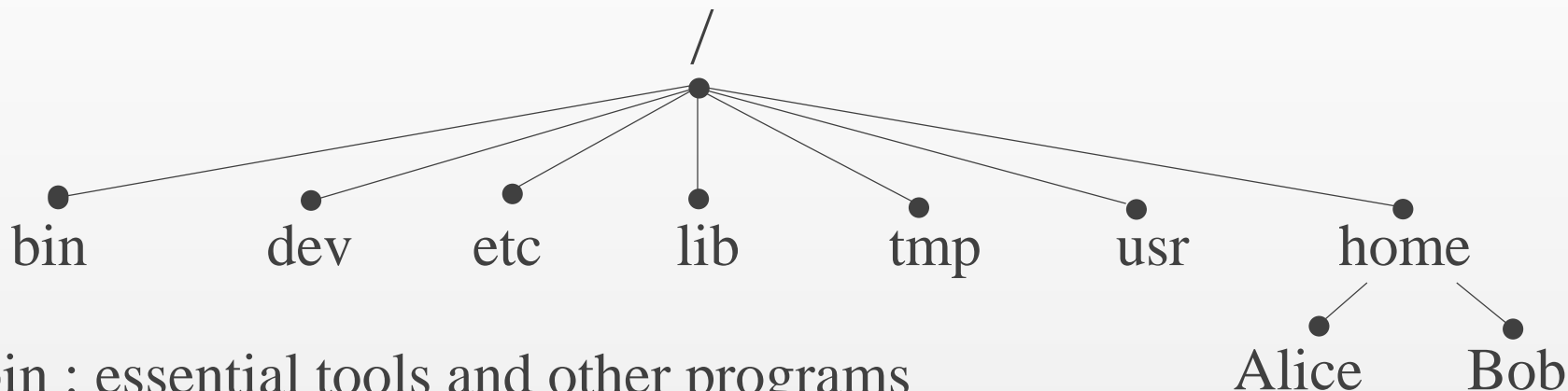
~ (user's directory i.e. /home/username)

relative path name      . (the current directory)

.. (the parent of the current directory )

# Basic Conceptions & Environment

## File System



`/bin` : essential tools and other programs

`/dev` : files representing the system's hardware devices

`/etc` : system configuration files

`/home` : the home directory for all system's users

`/lib` : essential system library files

`/proc` : files that give information about current system

`/usr` : files related to user tools and applications

# Basic Conceptions & Environment

## User & Group

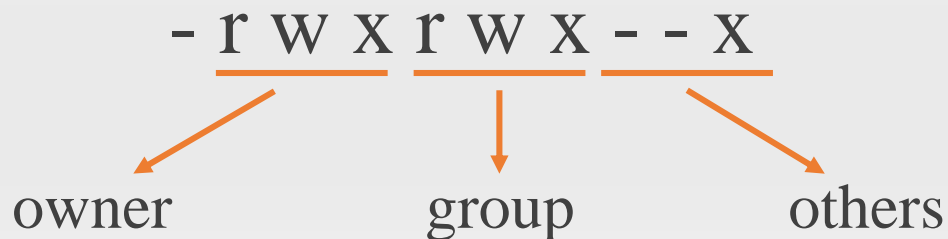
The system determines whether or not a **user** or **group** can access a file or directory.

There is a special user called **Super User** or the **root** which has permission to access any file and directory.

## Three Permissions

r – read      w – write      x – execute

Permissions for three categories of users



# Basic Conceptions & Environment

## Environment Variables

Environment variables are **a set of values** that can affect the way running processes will behave on a computer.

- **PATH** -- Contains a colon-separated list of directories that the shell searches for commands that do not contain a slash in their name.
- **HOME** -- Contains the location of the user's home directory.

.....

Set The Environment Variables:

**export VARIABLE = value**      (temporary)

**/etc/profile**      (permanent, for all users)

**.bash\_profile**      (permanent, for one user)

# Basic Conceptions & Environment

## Environment Variables

~/Desktop/Folder1/test.out -- a program to output “Hello Linux!”

```
user1@ubuntu:~/Desktop/Folder1$ ./test.out
```

Hello Linux!

```
user1@ubuntu:~/Desktop/Folder1$ cd ..
```

```
user1@ubuntu:~/Desktop$ ./test.out
```

test.out command not found

```
user1@ubuntu:~/Desktop$
```

**export**

```
PATH=$PATH:~/Desktop/Folder1
```

```
user1@ubuntu:~/Desktop$ ./test.out
```

Hello Linux!

## PART I

- Brief Introduction
- Basic Conceptions & Environment
- **Basic Commands**
- Shell Script

# Basic Commands

command [-options] [arguments]

Commands are often followed by **one or more options** that modify their behavior, and further, **by one or more arguments**, the items upon which the command acts.

- man      help      --help
- ls
- cd      mkdir      mvdir
- rm      mv      cp
- find      locate      grep
- cat
- >      >>      |      xarg
- sed      awk



# Basic Commands

man (manual)

provide a formal piece of **documentation** called a manual or man page.

```
$ man ls
```

help

similar to “man”, but more concise

```
$ help cd
```

--help

Display a description of the command's supported syntax and options

```
$ ls --help
```

# Basic Commands

## ls (list)

display a list of files and subdirectories

- a list all files, even those with names that begin with a period, which are normally not listed (i.e., hidden).
- l Display results in long format.

```
user1@ubuntu:~$ ls
```

```
Desktop Document Templates Downloads Public
```

```
user1@ubuntu:~$ ls -l /bin
```

-rwxr-xr-x	1	root	root	1021112	Oct 7
2014	bash				
permission	own	group	size (byte)	creating date	file (or dir.) name
	link number				
	(or files number)				

# Basic Commands

cd (changes directory)

`$ cd dir1` changes the working directory to “dir1”

`$ cd -` changes to the previous working directory

mkdir (make directory)

`$ mkdir ../dir1` Create the directory named “dir1”,  
if the path “...” exists.

`$ mkdir -p ../dir1` If the path “...” doesn’t exist,  
create each directories in it.

rmdir (remove **empty** directory)

`$ rmdir ../dir1` Remove a single directory named “dir1”,  
if it’s empty.

`$ rmdir -p ../dir1` Also remove the directories in the path  
“...”,

if they become empty

# Basic Commands

## rm (remove)

- r (--recursive) recursively delete directories.  
if a directory being deleted has subdirectories,  
delete them too.
- f (--force) ignore nonexistent files and do not prompt.

## mv (move)

**\$ mv [ -i ] file1 file2**      Move file1 to file2.

If file2 exists, it will be overwritten.

-i    prompt user before it is overwritten

**\$ mv dir1 dir2**      Move dir1 (and its contents) into dir2.

If dir2 does not exist, it will be created.

**\$ mv file1 dir1**      Move file1 into dir1. dir1 should  
already exist.

# Basic Commands

## cp (copy)

copy files or directories (similar to “mv”, but preserve the origin)

`$ cp [ -i ] file1 file2`      Copy file1 to file2.

If file2 exists, it will be overwritten.

-i    prompt user before it is overwritten

`$ cp -r dir1 dir2`      Copy dir1 (and its contents) into dir2.

If dir2 does not exist, it will be created.

`$ cp file1 dir1`      Copy file1 into dir1. dir1 should already exist.

`$ cp dir1/* dir2`      Copy all the files in dir1 into dir2.

---

## Wildcards

*	Matches any characters	ex*.jpg
---	------------------------	---------

?	Matches any single character	ex??-??-??.jpg
---	------------------------------	----------------

# Basic Commands

## find

searching for files or directories (files meeting specific criteria.)

```
$ find dir1 -name "*.jpg" -size +1M
```

finding any files whose name ending with “.jpg”

and size larger than 1M in dir1

-type -user -group ...

## find Logical Operators

-and (-a)            -or (-o)            -not (!)

```
$ find dir1 \( -name "*.png" \) -o \( -name "*.jpg" -a ! -  
user "root" \)
```

## locate (similar to “find -name”)

performs a rapid database search, **faster** than “find”

better to “updatedb” (update the database manually) before “locate”

# Basic Commands

grep (global regular expression print)

searches text files for the occurrence of a specified **regular expression** and outputs any **line** containing a match **to standard output**.

**\$ grep [-options] regex [file...]**

- i Ignore case.  
Do not distinguish between upper and lower case characters.
- l Print the name of each file that contains a match
- h For multi-file searches, suppress the output of filenames.

# Basic Commands

cat (concatenate)

read one or more files and copies them to **standard output**.

```
$ cat [file1...]
```

If cat is not given any arguments, it reads from standard input, by default, attached to the keyboard.

Type a <ctrl>+d to tell “cat” that it has reached end of file (EOF) on standard input.

```
$ cat
```

```
Hello World!  <ctrl>+d
```

```
Hello World!
```



# Basic Commands

› && >> (redirection)

`$ command1 > file1`

output

Change the destination of standard

`$ cat file1 file2 > file3`

into file3.

Concatenate file1 file2, and output

If file3 exists, it will be overwritten.

`$ cat file1 file2 >> file3`

destination,

The output will not overwrite the

but **attaching** to the back.

| (pipeline)

`$ command1 | command2`

command1 has standard output, and command2 has standard input.

`$ ls /bin /usr/bin | sort`

# Basic Commands

## xargs

It accepts input from standard input and converts it into an **argument list** for a specified command.

```
$ find /bin -name "a*" | list -l
```

×

```
$ find /bin -name "a*" | xargs list -l
```

✓

-a file            using file as the standard input

```
$ find /bin -name "a*" > file1.txt
```

```
$ xarg -a file1.txt list -l
```

-e 'flag'            set a separator ( ' ' or '\t' by default)

-n num              set the maximum number of arguments

# Basic Commands

sed (stream editor)

awk (Aho, Weinberg & Kernighan)

- powerful for text editing, especially for well-formed data.
- prefer to process the data as rows

sed      process the whole rows

awk      useful for well-formed data  
          able to process fields (columns) in rows  
          used like a programming language  
          do complex operations ( if else while for ... )

# Basic Commands

sed (stream editor)

awk (Aho, Weinberg & Kernighan)

Example :

```
user1  a
user2  undefined
user3  b
user4  c
user5  undefined
```

```
$ awk -F : '$2=="undefined" {printf("%s\n",$1)}'
```

```
user1
user2  user2
user3  b
user4  c
user5  user5
```

## PART I

- Brief Introduction
- Basic Conceptions & Environment
- Basic Commands
- **Shell Script**

# Shell Script

## Interactive VS. Shell Script

shell script -- a computer program designed to be run  
(interpretive execution) by the shell.

- convenient: reusable
- capable: variables, branches, loops...

a script file with filename extension “.sh”

```
#!/bin/bash
```

```
.....
```

```
.....
```

run a script

```
$ chmod 777 ????.sh
```

```
$ ./???.sh
```

```
$ bash ./???.sh
```

# Shell Script

## Variables

Define, Assignment & Read

**VariableName=value**

**read VariableName**

- no space between VarName and the equality sign
- first letter: a-z A-Z
- no keywords of shell

Use a variable

**\$VariableName**

\$0	filename of the script
\$n	the n-th argument
\$#	the number of the arguments
\$HOME	user directory
\$\$	pid

Some System Variable

# Shell Script

## Variables

Example :

test.sh

```
#!/bin/bash
```

```
read a
```

```
read b
```

```
c=$((a+b)**a)
```

```
echo $c
```

```
$ chmod 777 ./test.sh
```

```
./test.sh
```

```
2
```

```
3
```

```
25
```

It will output  $(2+3)**3$  if without `$( )`

using arguments

```
#!/bin/bash
```

```
echo $((($1+$2)**$1)) ./test.sh 2 3
```



# Shell Script

## String

single quotes

```
str='no variables or escape character'
```

double quotes

```
v='variables'
```

```
str="$v or \"escape character\""
```

connecting

```
str1="connecting strings"
```

```
str2="simple"
```

```
str3=$str1 " is " $str2
```

length        \${#string}

substring     \${#string:begin:end}

# Shell Script

## Printf

`printf format-string [arguments...]`

Different from “printf” in C

- no ( )
- using space between two arguments

if the number of arguments is **greater than** the number of % in format,

The format-string will be **reused** repeatedly

```
printf “%s %s\n” 1 2 3 4
```

```
1 2
```

```
3 4
```

# Shell Script

## Branches

```
if [ condition ]  
then  
    ...  
else  
    ...  
fi
```

---

```
if [ condition1 ]; then  
    ...  
elif [ condition2 ]; then  
    ...  
else  
    ...  
fi
```

Operator	Remark
-eq	==
-ne	!=
-gt	>
-lt	<
-ge	>=
-le	<=

### Numerical Comparison Operator

Operator	Remark
=	== for string
!=	!= for string
-z	if a string is empty
-f / -d	is file / is dir.
-r / -w / -x	check permission
-e	if a file/dir. exists

### Other Operator

# Shell Script

## Loops

```
for variable in list
do
    ...
done
```

---

```
while [ condition ]
do
    ...
done
```

---

```
break loop_num
continue loop_num
```

```
for FILE in $HOME/*
do
    echo $FILE
done
```

```
count=0
while [ $count -lt 5 ]
do
    count=$((count+1))
    echo $count
done
```

## PART II

- Text Editor (Vim)
- Compile & Debug (for C)

# Text Editor (Vim)

- Vim's interface is **not** based on menus or icons, but **on commands** given in a text user interface.

Intall

edit & update the sources

edit the source list file: `/etc/apt/sources.list`

`$ sudo apt-get update`

**Super User Do**

**Advanced Package Tool**

install vim

`$ sudo apt-get install vim`

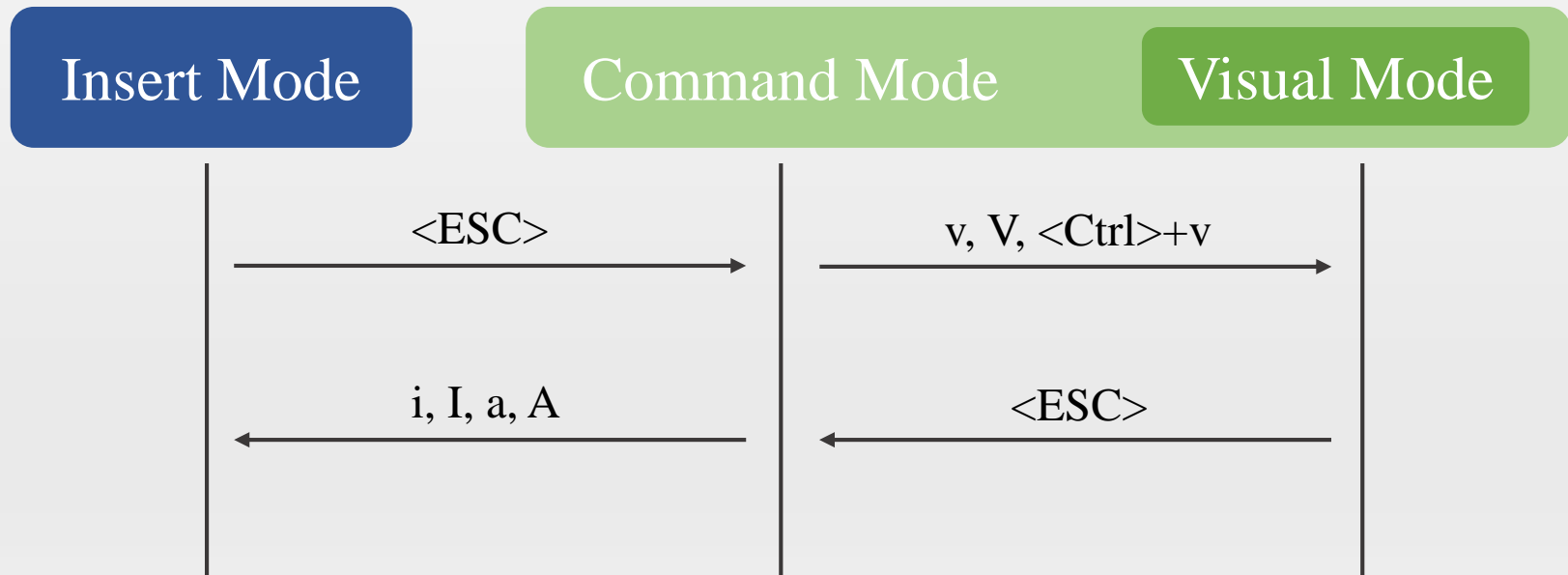
obtain a vim's tutorial

`$ vimtutor`

# Text Editor (Vim)

## Three Modes

- Command mode: all keystrokes are interpreted as commands
- Insert mode: most keystrokes are inserted as text
- Visual mode: helps to visually select some text, may be seen as a submode of the the command mode.



# Text Editor (Vim)

## Quit and Save

<b>w</b>	write the current buffer to disk (save)
<b>q</b>	close the current window
<b>x</b>	save and close
<b>q!</b>	close without save

## Scroll the Screen

<b>&lt;Ctrl&gt;+f</b>	1 page
<b>&lt;Ctrl&gt;+d</b>	1/2 page
<b>&lt;Ctrl&gt;+e</b>	1 line
<b>&lt;Ctrl&gt;+y</b>	1 line
<b>&lt;Ctrl&gt;+u</b>	1/2 page
<b>&lt;Ctrl&gt;+b</b>	1 page





# Text Editor (Vim)

## Movement of the Cursor

<b>j</b> = ↑	<b>k</b> = ↓	<b>h</b> = ←	<b>l</b> = →
<b>0</b>	first column of the line		
<b>^</b>	first non-blank character of the line		
<b>w</b>	jump to next word		
<b>W</b>	jump to next word, ignore punctuation		
<b>e</b>	jump to word-end		
<b>E</b>	jump to word-end, ignore punctuation		
<b>b</b>	jump to word-beginning		
<b>B</b>	jump to word-beginning, ignore punctuation		
<b>ge</b>	jump to previous word-ending		
<b>gE</b>	jump to previous word-ending, ignore punctuation		
<b>g_</b>	jump to last non-blank character of the line		
<b>\$</b>	jump to the last character of the line		
<b>%</b>	jump to the matching bracket		

# Text Editor (Vim)

## Editing

- d** delete the characters **from the cursor position to the position given by the next command (FCTN)**
- c** cut the character FCTN
- x** delete the character **under** the cursor
- X** delete the character **before** the cursor
- y** copy the characters FCTN
- p** paste previous deleted or copied text **after** the current cursor position
- P** paste previous deleted or copied text **before** the current cursor position
- r** replace the current character with the newly typed one
- s** substitute the text FCTN with the newly typed one
- .** repeat the last insertion or editing command

Doubling d , c or y operates on the whole line.

# Text Editor (Vim)

## Visual Block

**<Ctrl>+v** enter the visual block mode

selected a rectangle of text:

- i** insert text in front of it (switch to insert mode)
- a** insert text after it
- c** insert text to replace it

operates on the multiple columns

#incl <u>d</u> e <stdio.h>	$\xrightarrow{\text{type the command "a"}}$ then type character "u"	#include <stdio.h>
#incl <u>d</u> e <stdlib.h>		#include <stdlib.h>
#incl <u>d</u> e <math.h>		#include <math.h>

- Completion
- Searching & Replacing
- Marks

## PART II

- Text Editor (Vim)
- **Compile & Debug (for C)**

# Compile & Debug (for C)

## Compilation & Execution

GCC (GNU C Compiler) → (GNU Compiler Collection)

`$ gcc test.c` compile the C source file

produce an executable file named (by default) a.out

`$ ./a.out` run the program

## Useful Flags(Options)

`$ gcc -o TEST test.c` to specify the executable file's name

`$ gcc -Wall test.c` gives much better warnings

`$ gcc -g test.c` to enable debugging with gdb

`$ gcc -O test.c` to turn on optimization

# Compile & Debug (for C)

## Linking with Libraries

### Library

static version                      **lib+name.a**                      **(-static)**

dynamic version                      **lib+name.so**                      **(default)**

which can be found in the functions' or libraries' man page

some library routines do not reside in the C library

**-l+name**                      link with libraries manually

If the system can not find the library file in the default directory  
( /usr/local/lib/ & /usr/lib )

**-L+lib's dir**                      give the directory manually

# Compile & Debug (for C)

## Separate Compilation

compile a program with several separate files

```
$ gcc -c test1.c
```

```
$ gcc -c test2.c
```

```
...
```

```
$ gcc -c -o TEST test1.o test2.o ...
```

-c        compile to produce an object file, which is not executables  
          just machine-level representations of the source code

# Compile & Debug (for C)

## Makefiles

build the program **automatelly** according to the **makefile**

Makefiles are based on rules as:

```
target: prerequisite1 prerequisite2 ...  
    command1  
    command2  
    ...
```

Complie the Program

( test1.c & test2.c )

**\$ make**

**\$ make clean**

```
TEST: test1.o test2.o  
    gcc -o TEST test1.o test2.o  
test1.o: test1.c  
    gcc -c test1.c  
test2.o: test2.c  
    gcc -c test2.c  
clean:  
    rm -f test1.o test2.o
```

makefile



# Compile & Debug (for C)

## Debugging with GDB (GNU debugger)

**\$gdb** enter the gdb environment

Command	Remark
file <file name>	load a executable file
r	run
c	continue
b <line number> b <function name>	set Breakpoint
s, n	excute a line of source code
p <variable name>	print the value of a variable
q	quit

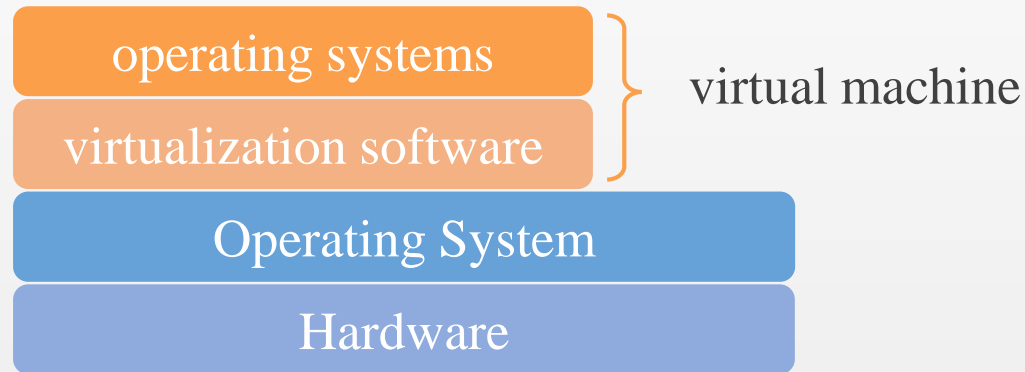
## PART III

- Install & Configure a Virtual Machine

# Install & Configure the Virtual Machine

## Virtual Machine

a virtual machine is an emulation of a particular computer system



## Virtualization Software

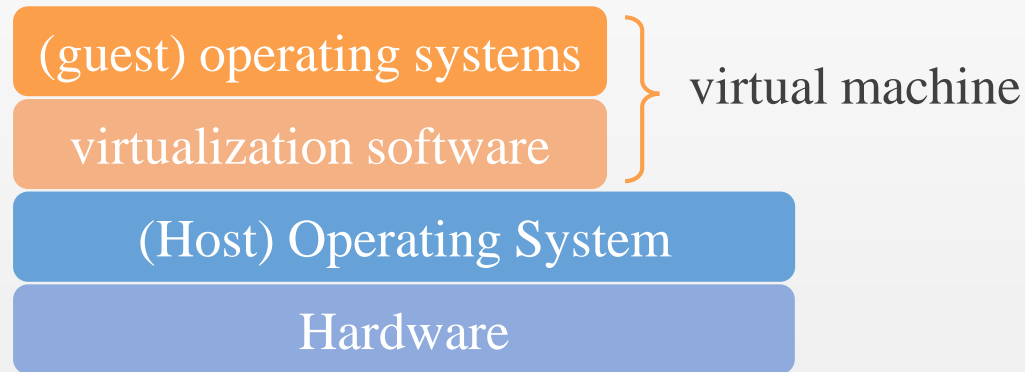
provide (hardware) resources virtually to the new OS

- VMware
- Virtual Box
- Virtual PC

# Install & Configure the Virtual Machine

## Virtual Machine

a virtual machine is an emulation of a particular computer system



## Virtualization Software

provide (hardware) resources virtually to the new OS

- VMware
- Virtual Box
- Virtual PC

# Install & Configure a Virtual Machine

## Install the Virtual Machine

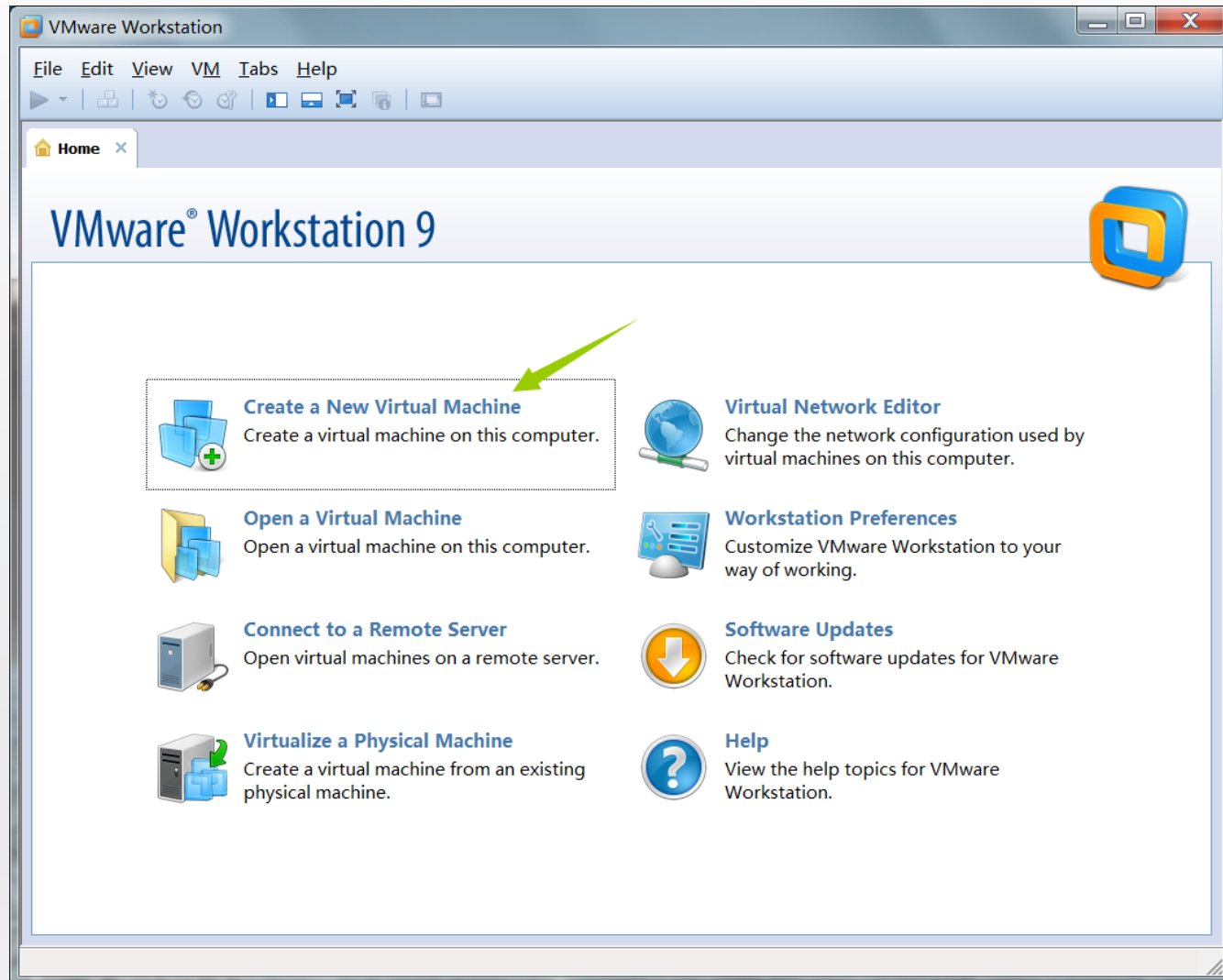
VMware Workstation 9.0 + Ubuntu 14.04 LTS ( kernel 3.19 )



- Download the Setup File of VMware 9.0
- Download the Ubuntu 14.04 LTS from the official website [www.ubuntu.com/download/desktop](http://www.ubuntu.com/download/desktop)
- Install VMware 9.0
- Create a Virtual Machine in the VMware

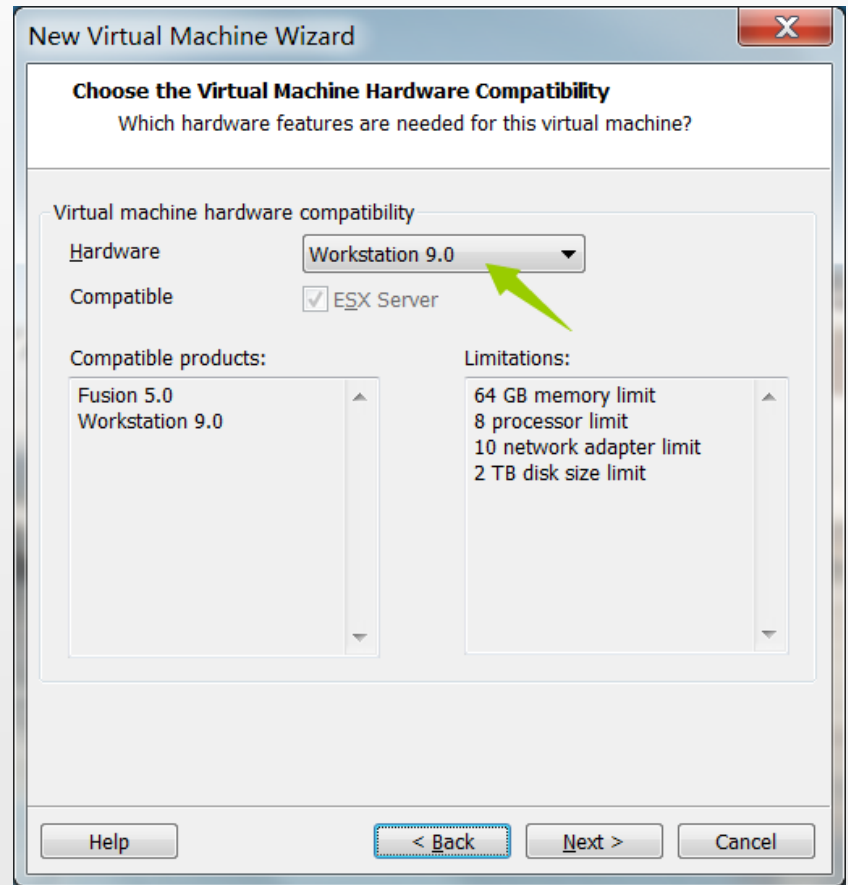
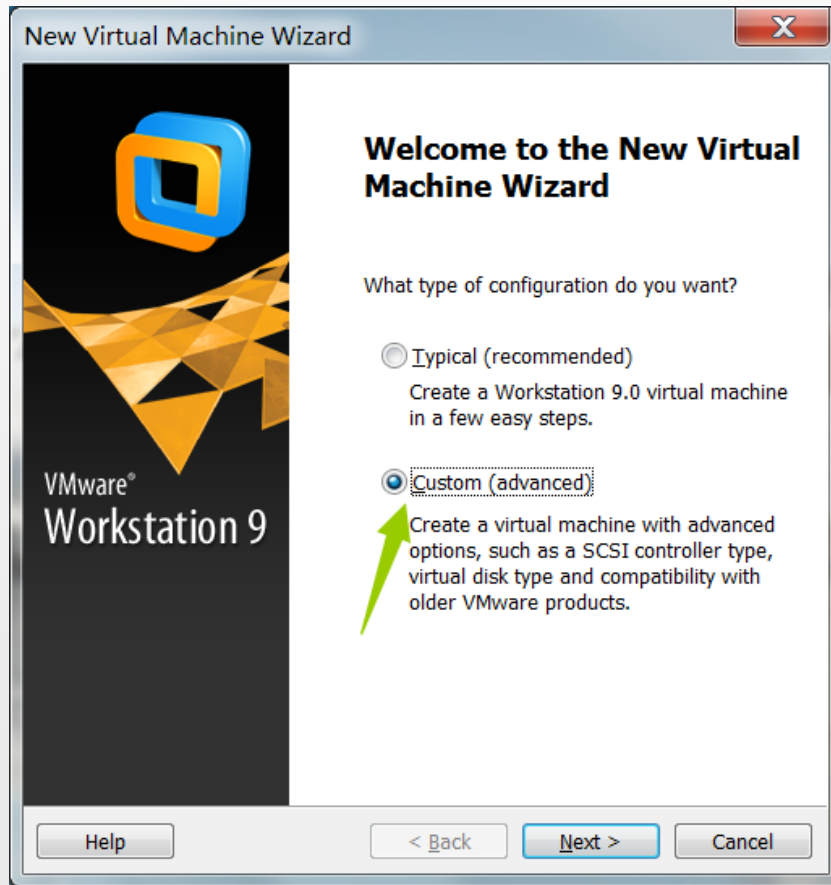
# Install & Configure a Virtual Machine

## Create a Virtual Machine



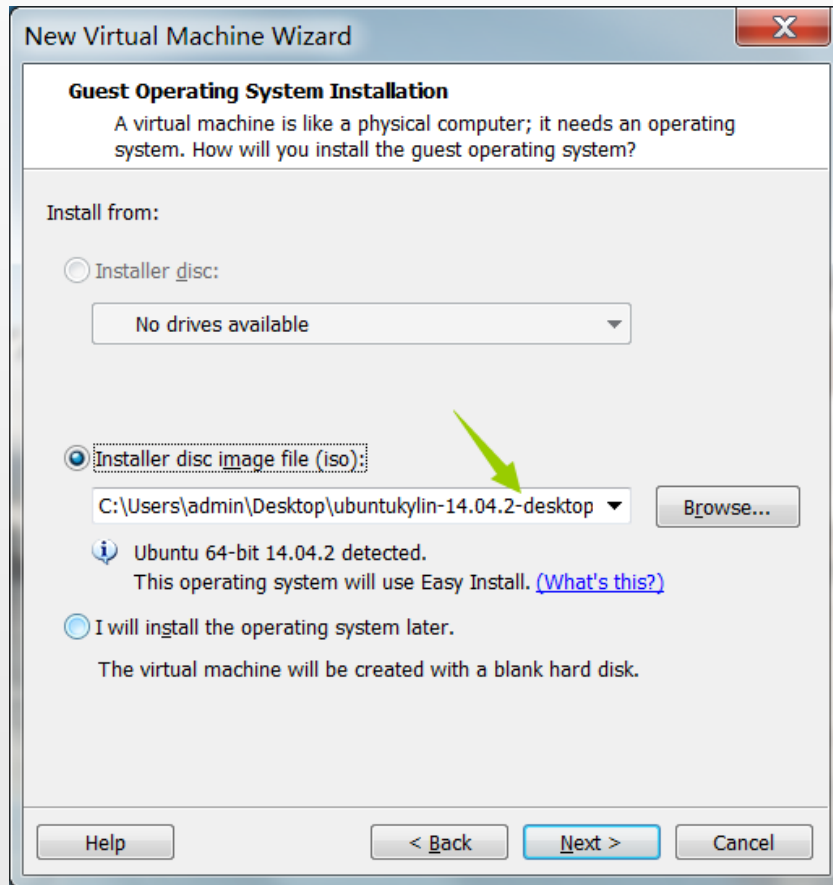
# Install & Configure a Virtual Machine

## Create a Virtual Machine

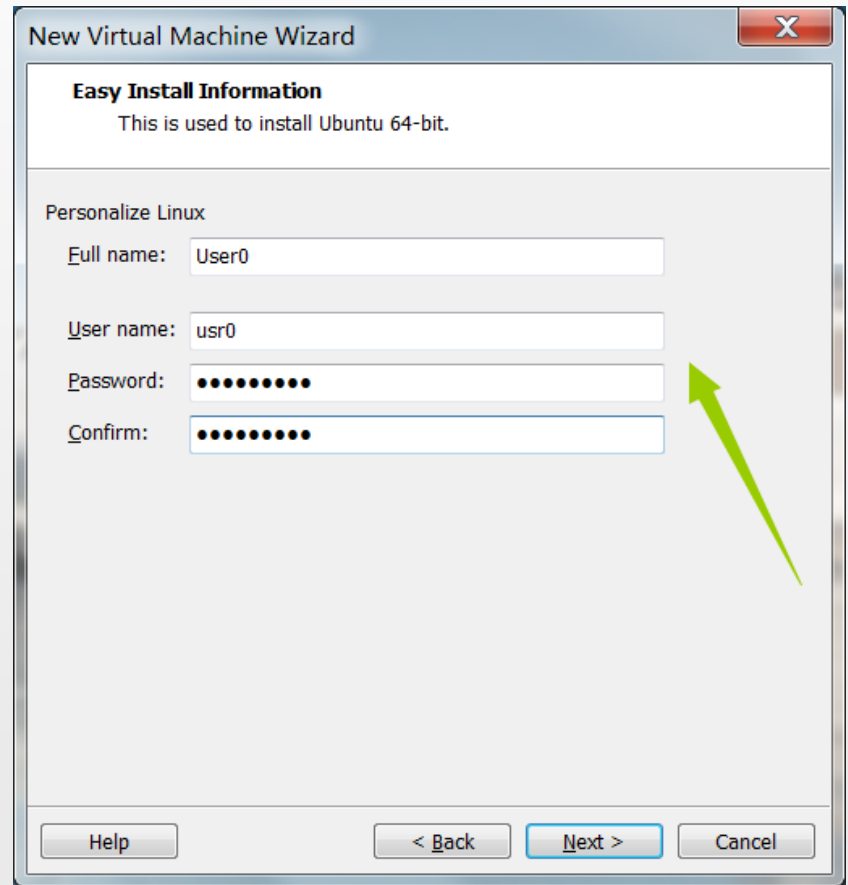


# Install & Configure a Virtual Machine

## Create a Virtual Machine



Select the .iso file of ubuntu  
downloaded before

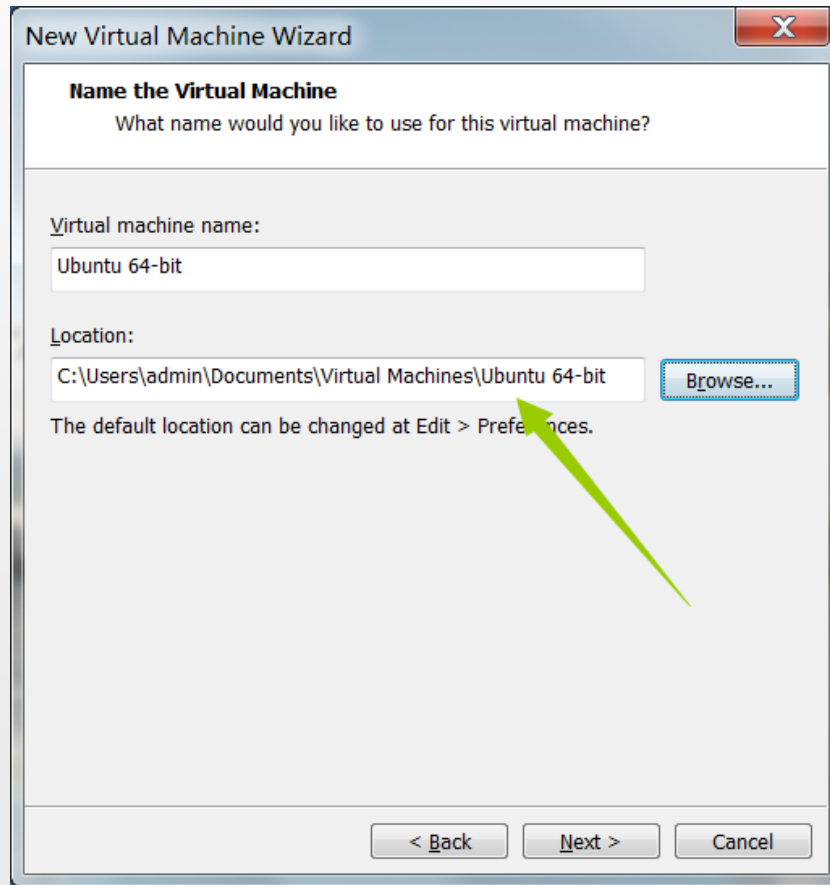


Fill the user name and  
the password of the super user



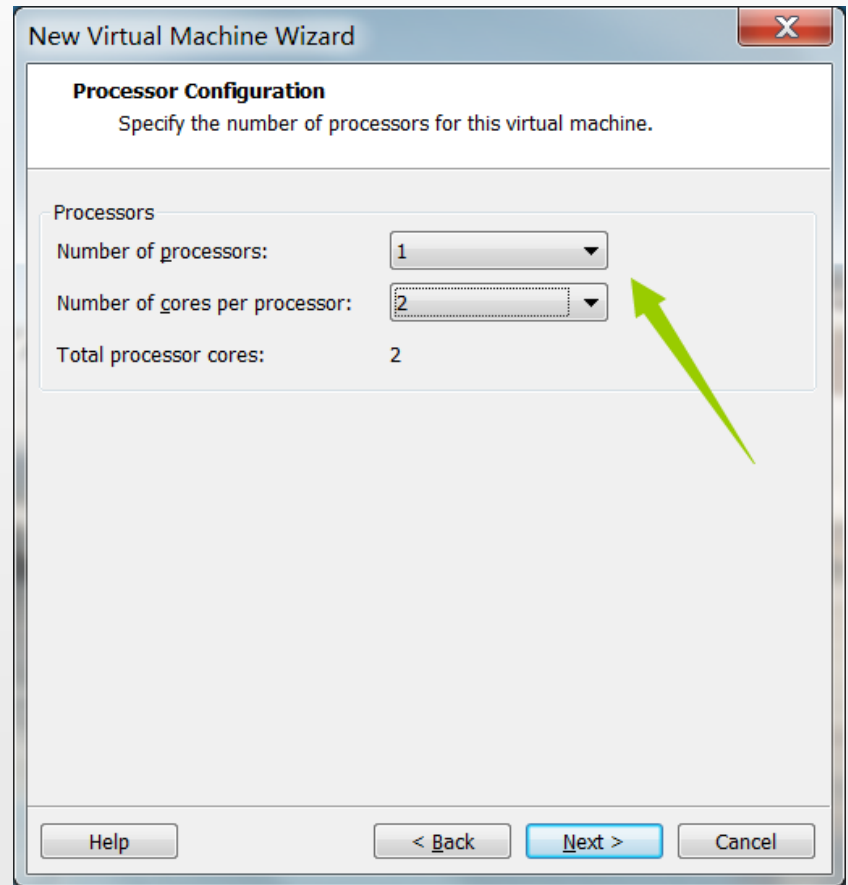
# Install & Configure a Virtual Machine

## Create a Virtual Machine



The screenshot shows the 'Name the Virtual Machine' step of the wizard. The title bar reads 'New Virtual Machine Wizard'. The main heading is 'Name the Virtual Machine' with the instruction 'What name would you like to use for this virtual machine?'. There are two input fields: 'Virtual machine name:' containing 'Ubuntu 64-bit' and 'Location:' containing 'C:\Users\admin\Documents\Virtual Machines\Ubuntu 64-bit'. A 'Browse...' button is next to the location field. A green arrow points to the location field. At the bottom are '< Back', 'Next >', and 'Cancel' buttons.

Fill the VM's name and  
select the location of the VM

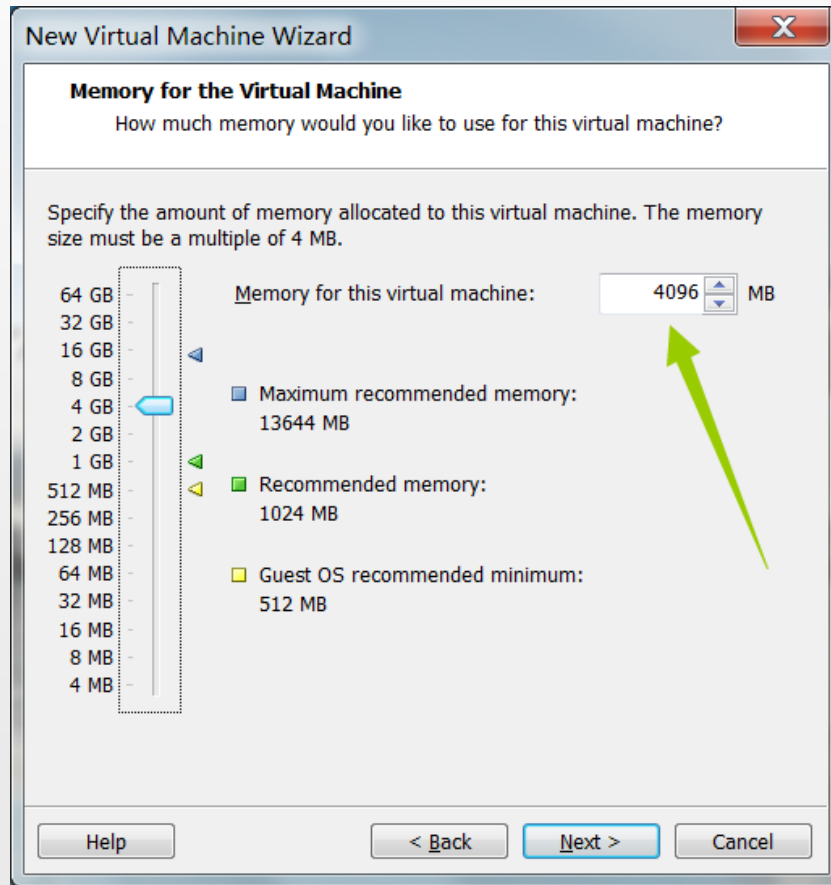


The screenshot shows the 'Processor Configuration' step of the wizard. The title bar reads 'New Virtual Machine Wizard'. The main heading is 'Processor Configuration' with the instruction 'Specify the number of processors for this virtual machine.'. Under the 'Processors' section, there are two dropdown menus: 'Number of processors:' set to '1' and 'Number of cores per processor:' set to '2'. A green arrow points to the 'Number of cores per processor' dropdown. Below these is the text 'Total processor cores: 2'. At the bottom are 'Help', '< Back', 'Next >', and 'Cancel' buttons.

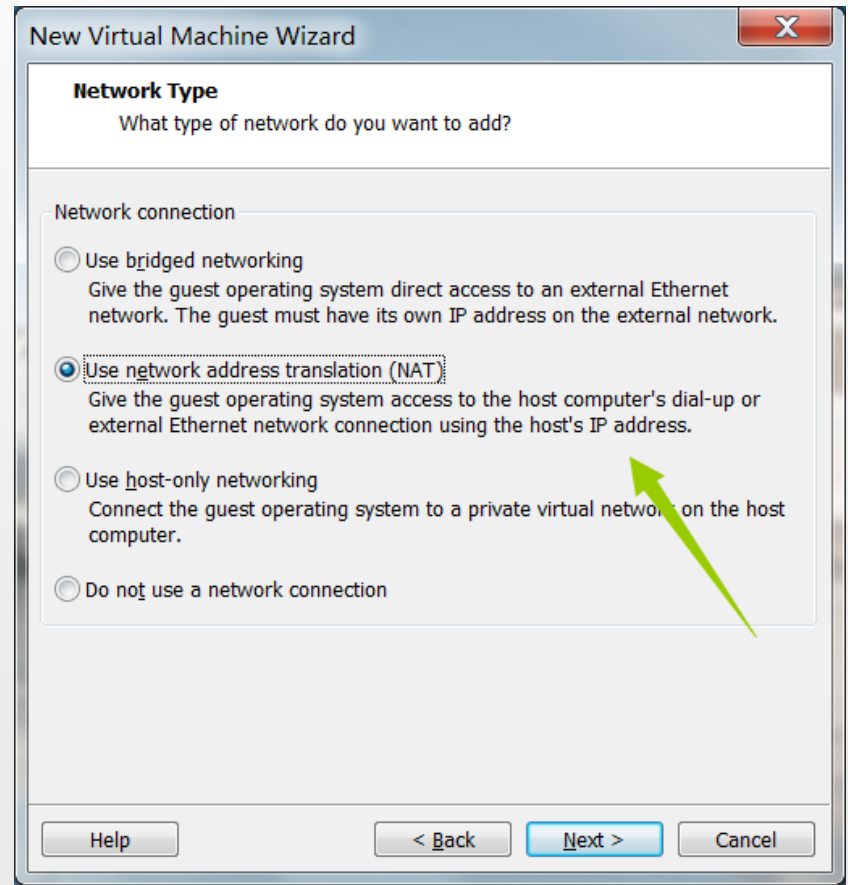
Set the number of processors for VM  
1 processor 1 cores are enough

# Install & Configure a Virtual Machine

## Create a Virtual Machine



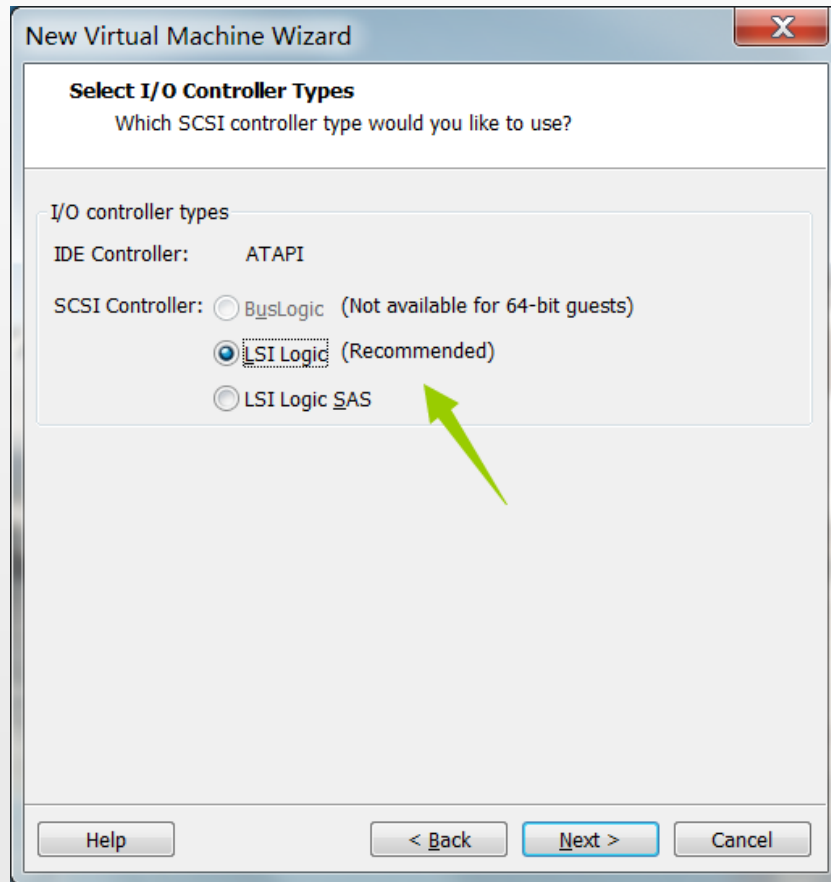
Set the memory for VM  
more than 1024MB



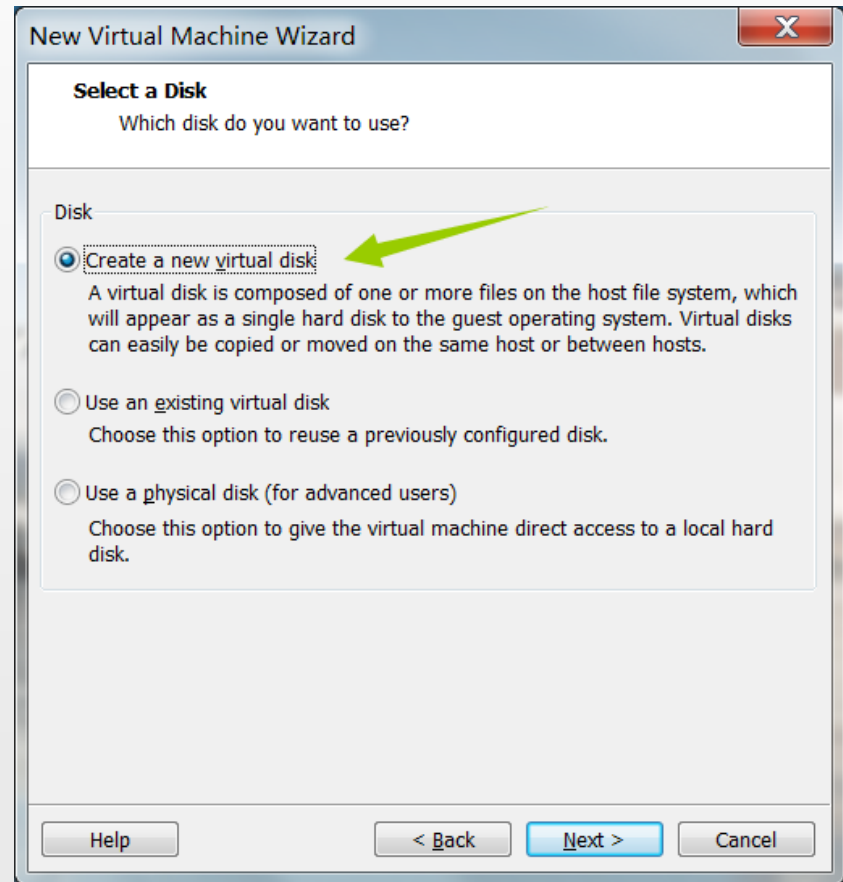
Set the network type  
NAT

# Install & Configure a Virtual Machine

## Create a Virtual Machine



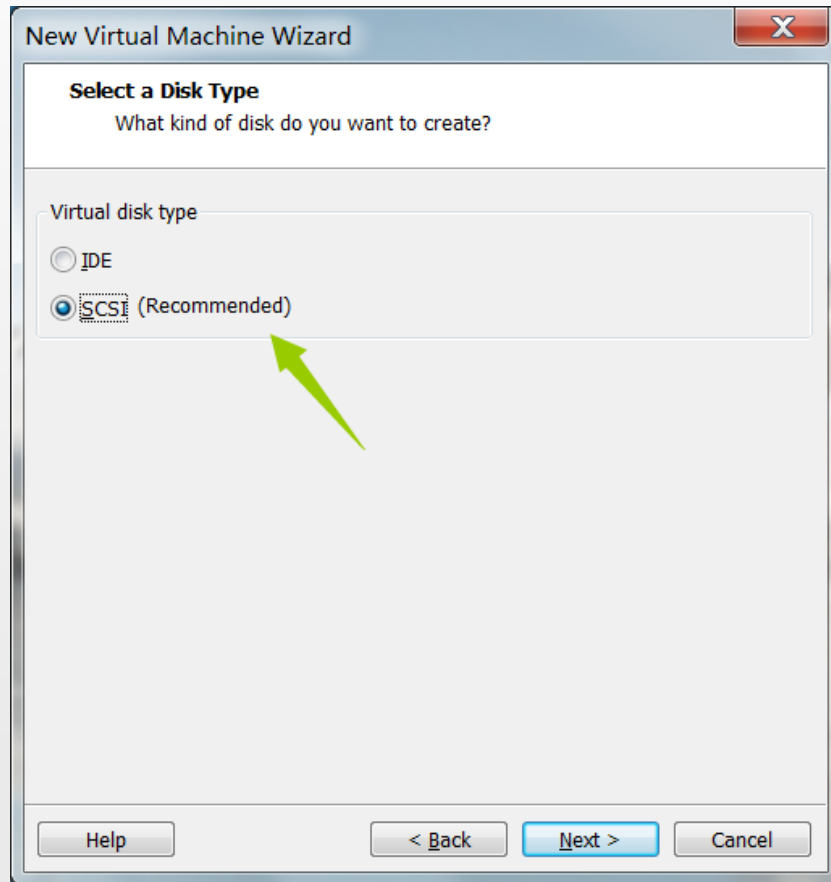
Set the I/O controller type  
default



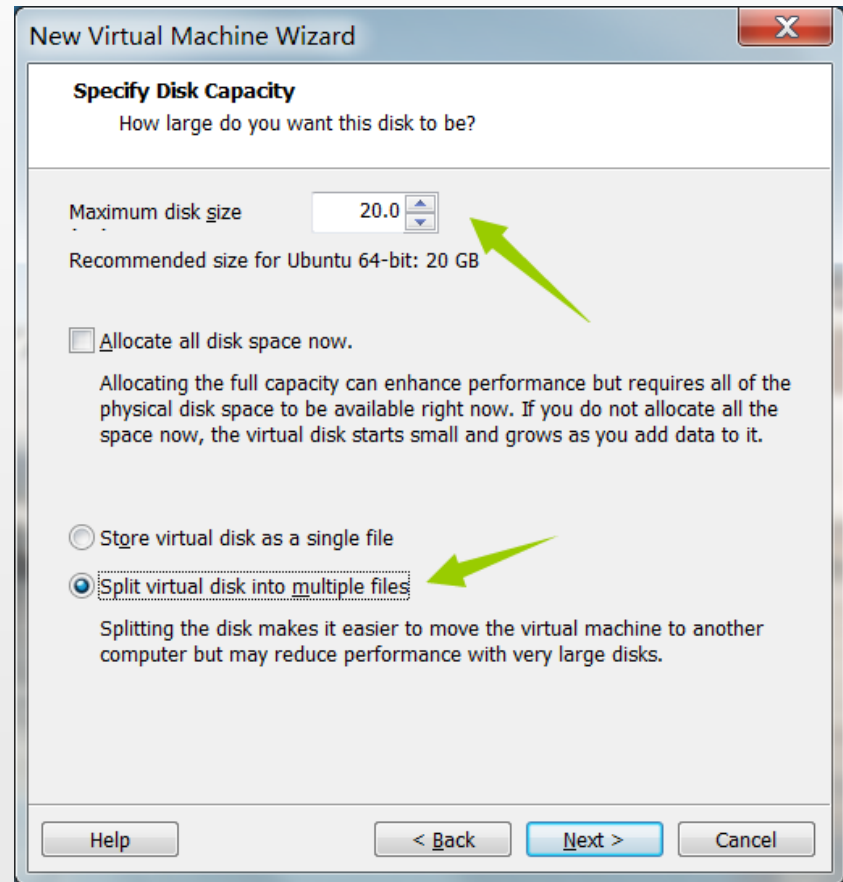
Create a virtual disk  
composed of files on host OS

# Install & Configure a Virtual Machine

## Create a Virtual Machine



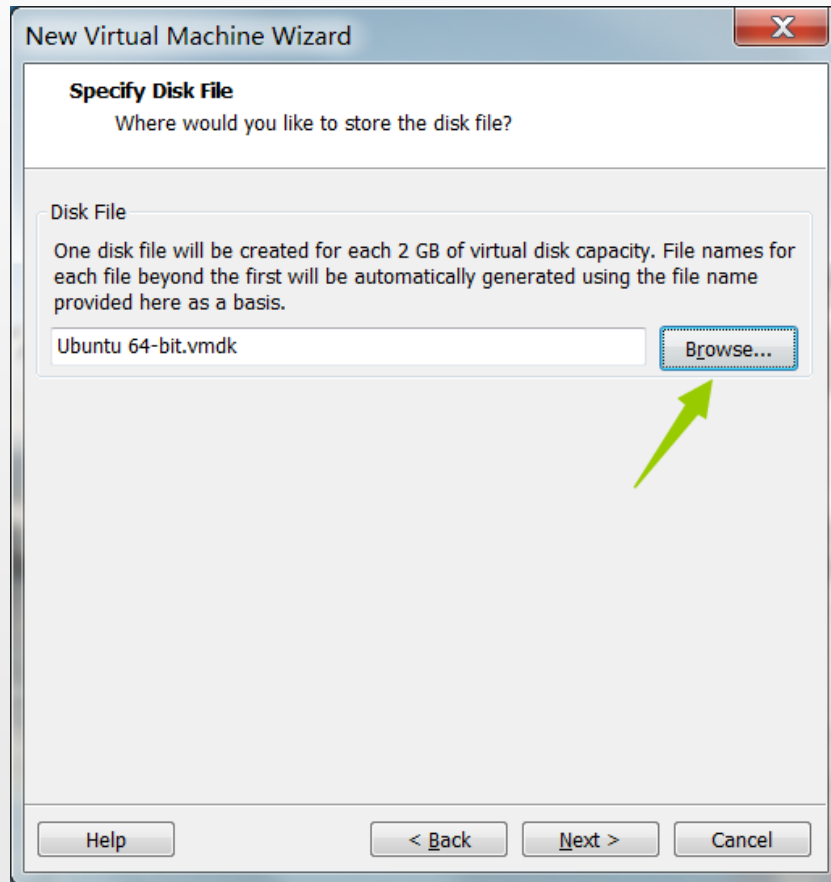
Set the disk type  
default



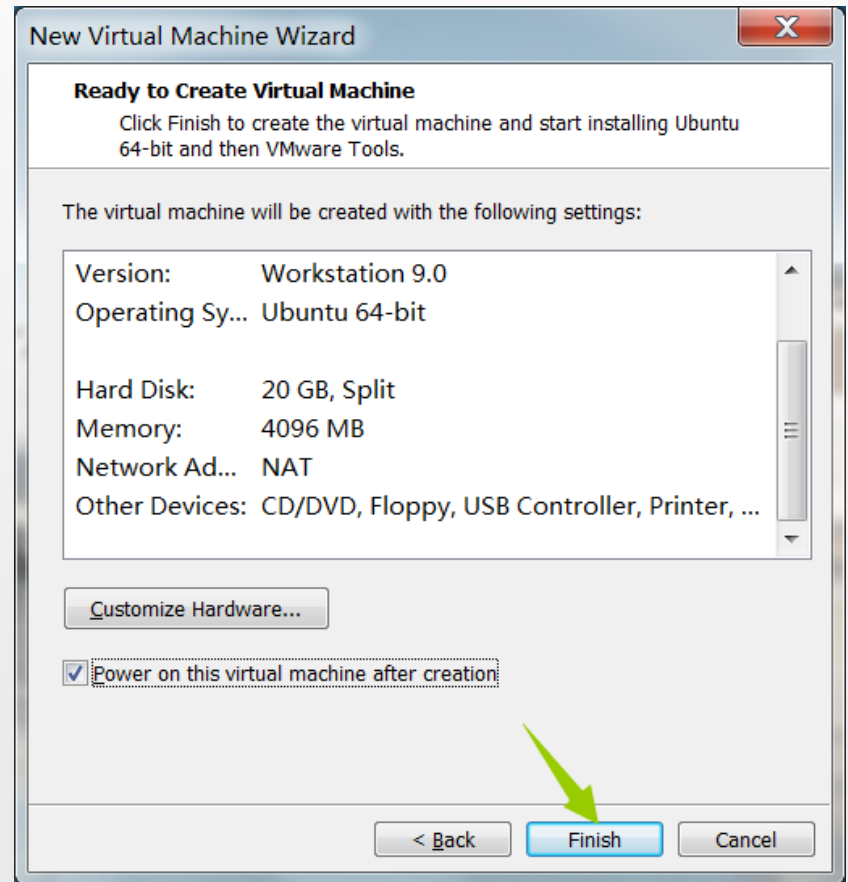
disk size: 15GB is enough  
Split into multiple files: easy to move

# Install & Configure a Virtual Machine

## Create a Virtual Machine



Select the location of disk files



Finish