

Introduction of Linux

Qiuyang LIU
Hanwei ZHANG
Dept. of Comput. Sci. &
Tech.
East China Normal
University

PART I

- Brief Introduction
- Basic Conceptions & Environment
- Install & Configure a Virtual Machine
- Basic Commands

PART II

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

PART I

- **Brief Introduction**
- Basic Conceptions & Environment
- Install & Configure a Virtual Machine
- Basic Commands

Brief Introduction

Linux (/'lɪnəks/)

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

1987
MINIX OS (for education)
Andrew S. Tanenbaum



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

1970s
BSD (Berkeley Software Distribution)

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**
- Install & Configure a Virtual Machine
- Basic Commands

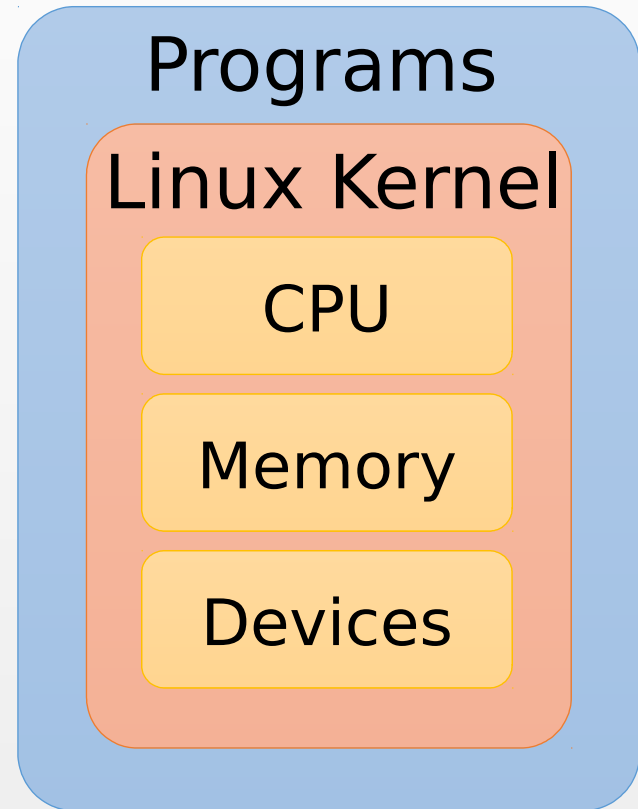
Basic Conceptions & Environment

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

.....

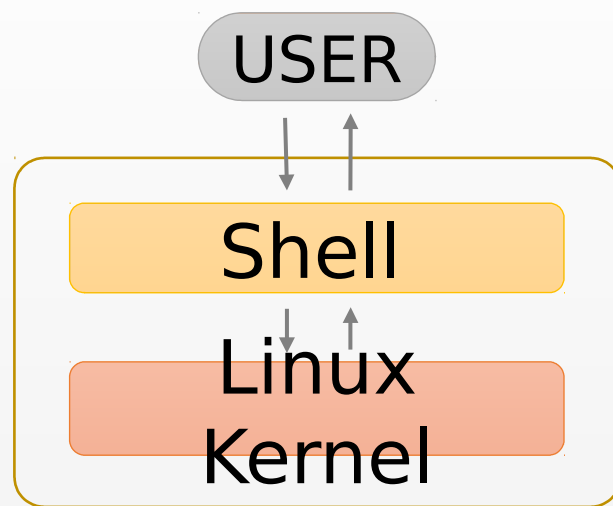


Basic Conceptions & Environment

Shell (CLI shell)

Command Line Interface

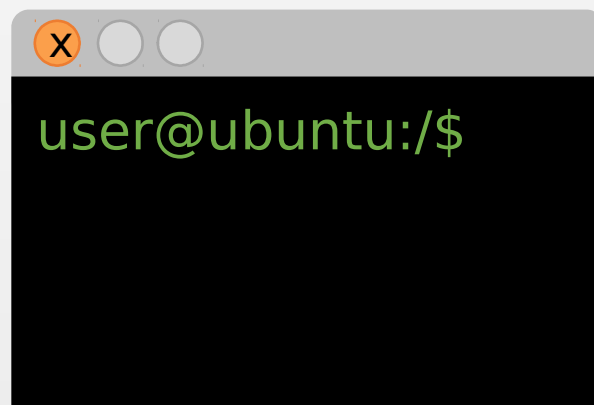
A **program** which accepts commands as text input and **converts commands** to appropriate operating system functions.



Terminal ↔ Shell

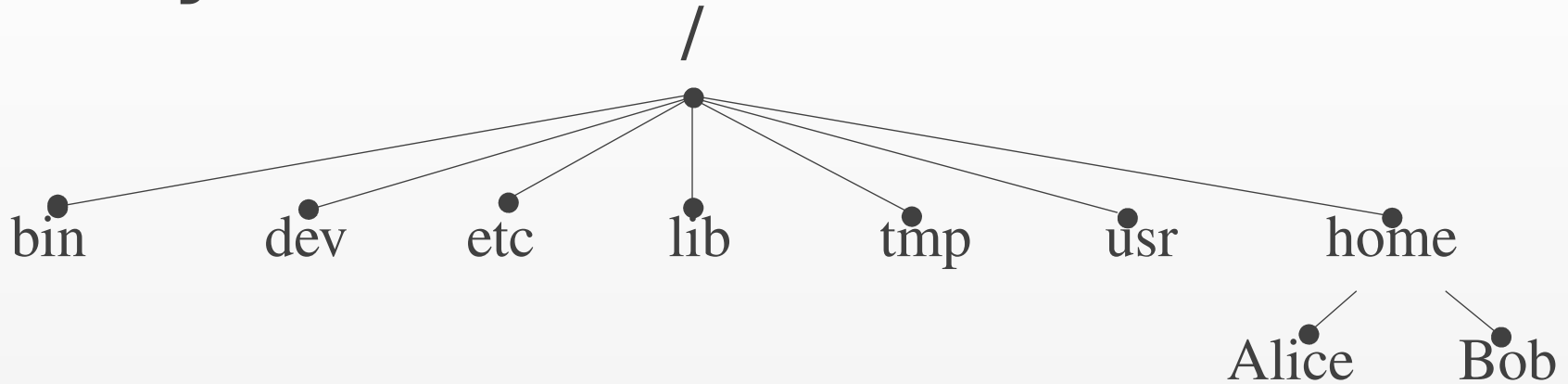
The terminal send information to the shell,

receive and display the information from the shell.



Basic Conceptions &

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 /)

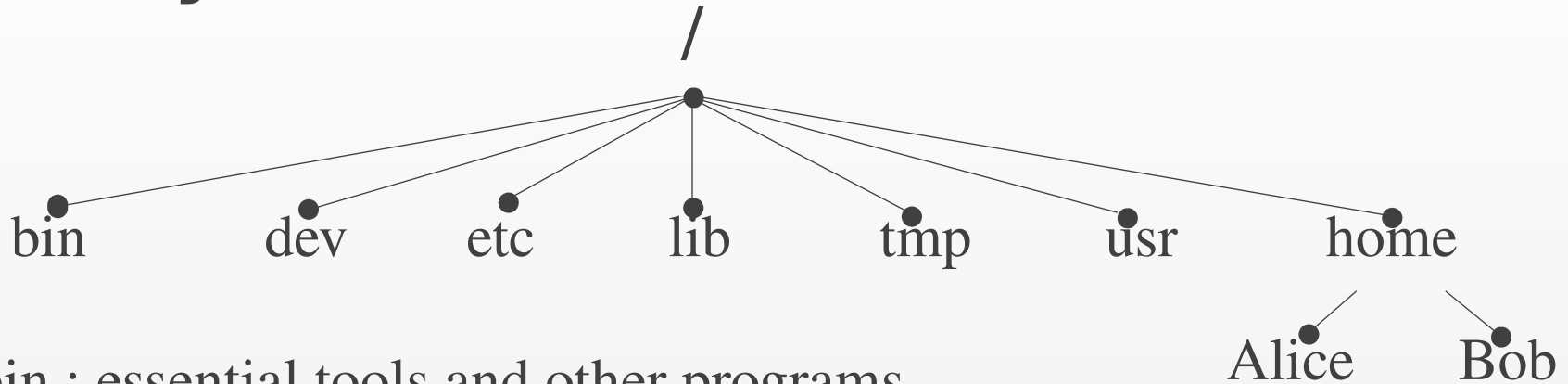
~ (user's directory i.e. /home/username)

relative path name . (the current directory)

.. (the parent of the current directory)

Basic Conceptions &

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 &

User & Group

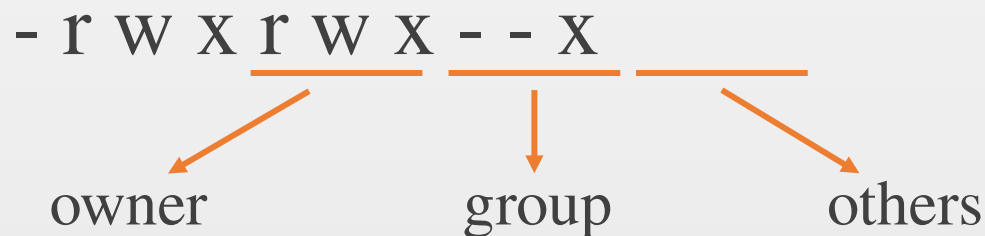
The system determines whether or not a **user** or **group** can access a file or directory.

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 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)

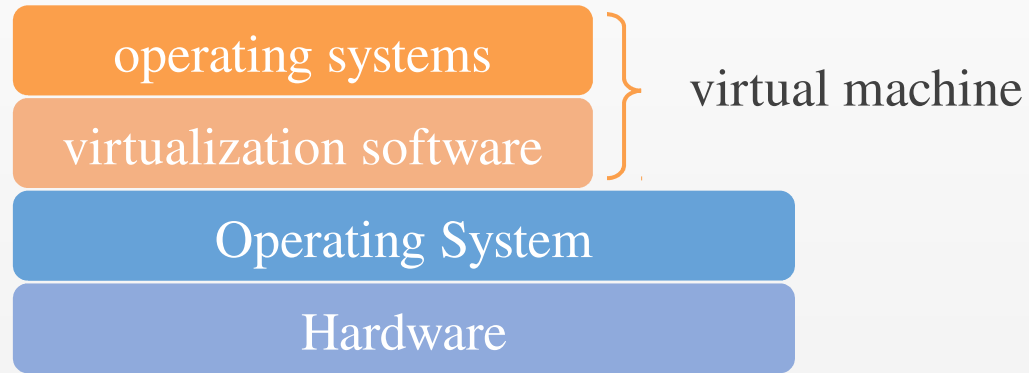
PART I

- Brief Introduction
- Basic Conceptions & Environment
- **Install & Configure a Virtual Machine**
- Basic Commands

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

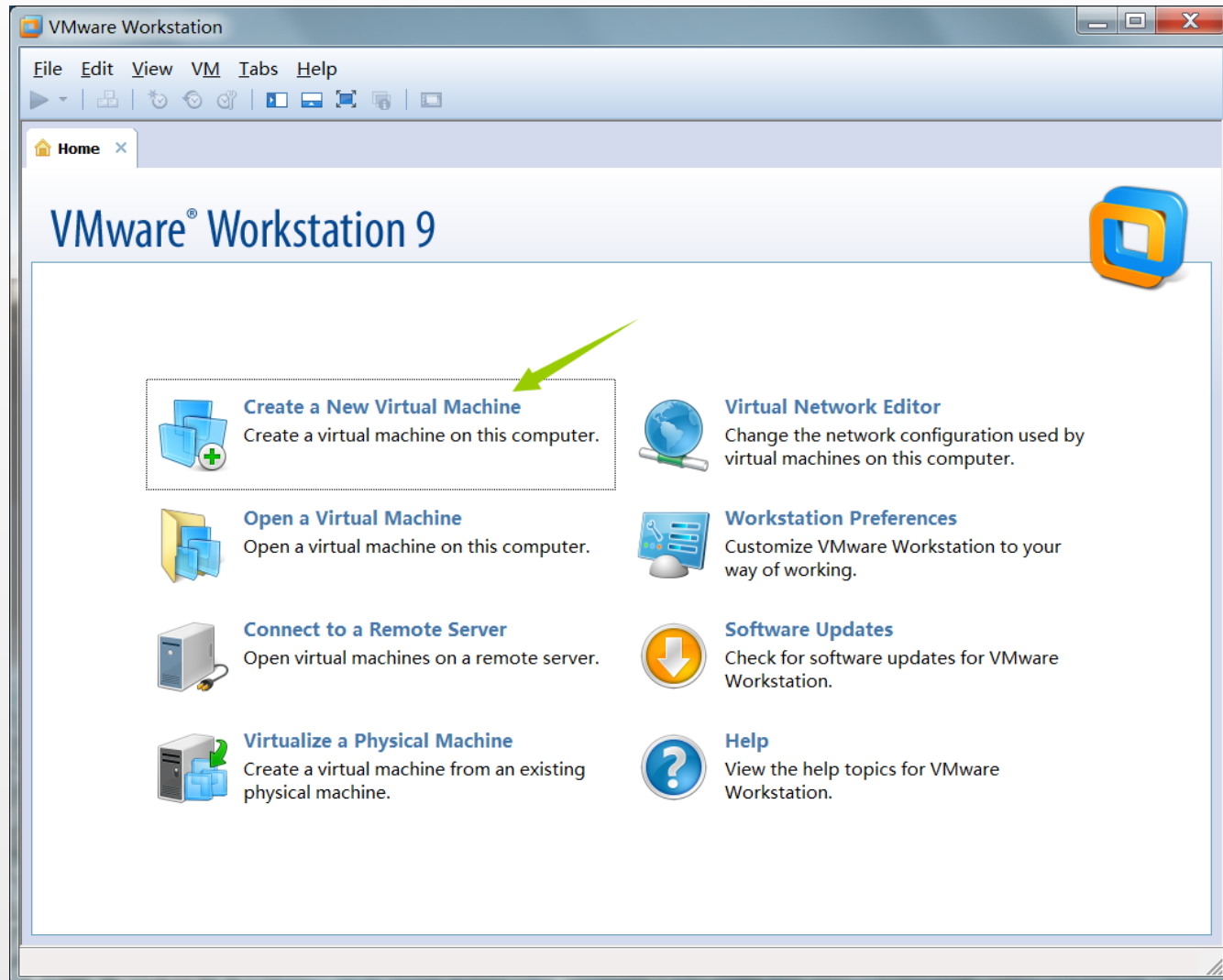
VMware Workstation 9.0 + Ubuntu 14.04 LTS (kernel 3.19)



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

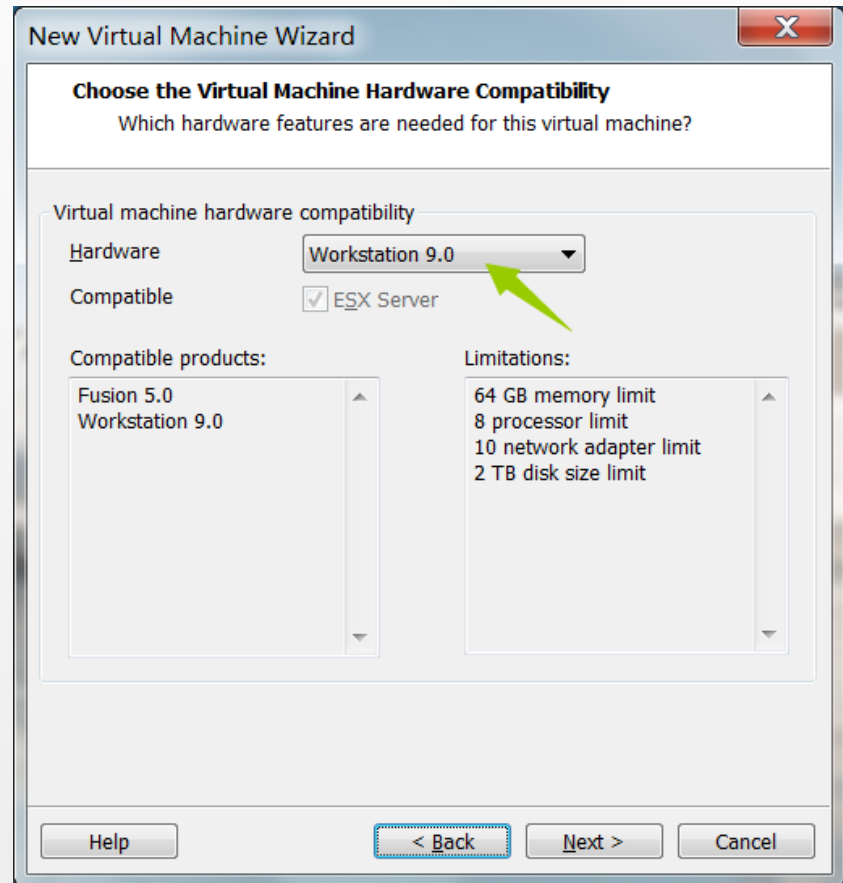
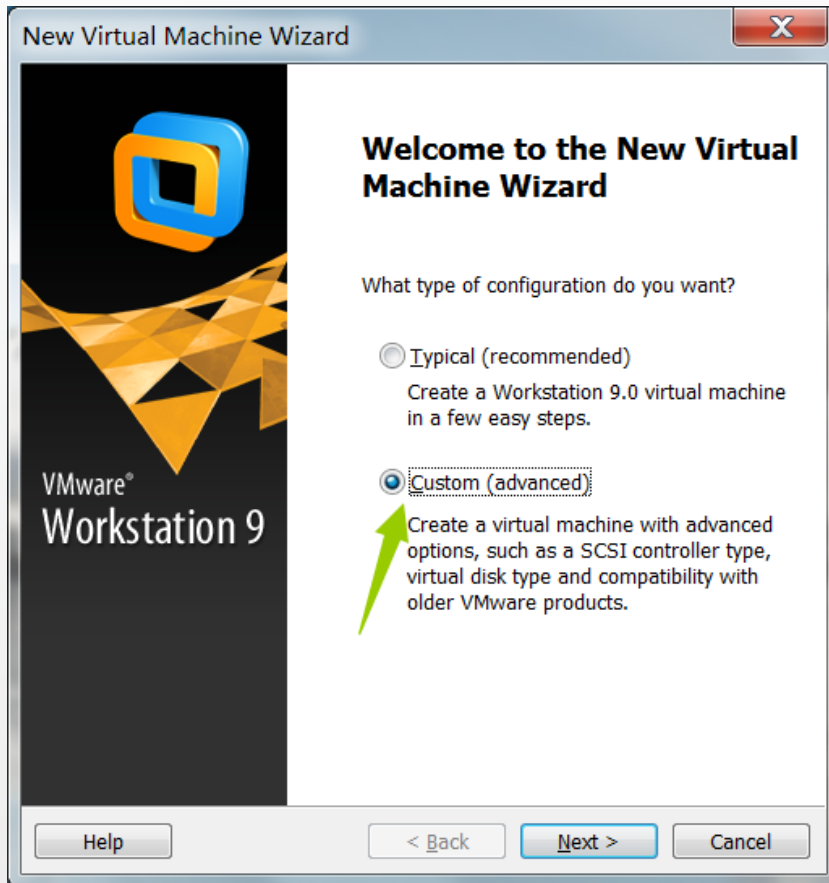
Install & Configure a Virtual

Create a Virtual Machine



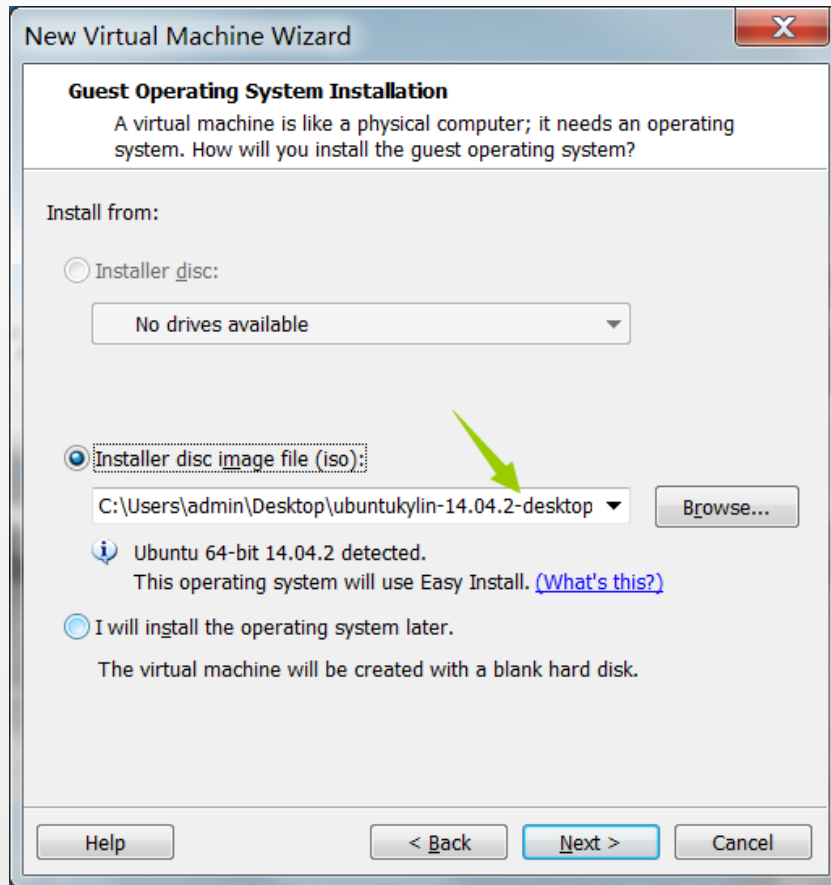
Install & Configure a Virtual

Create a Virtual Machine

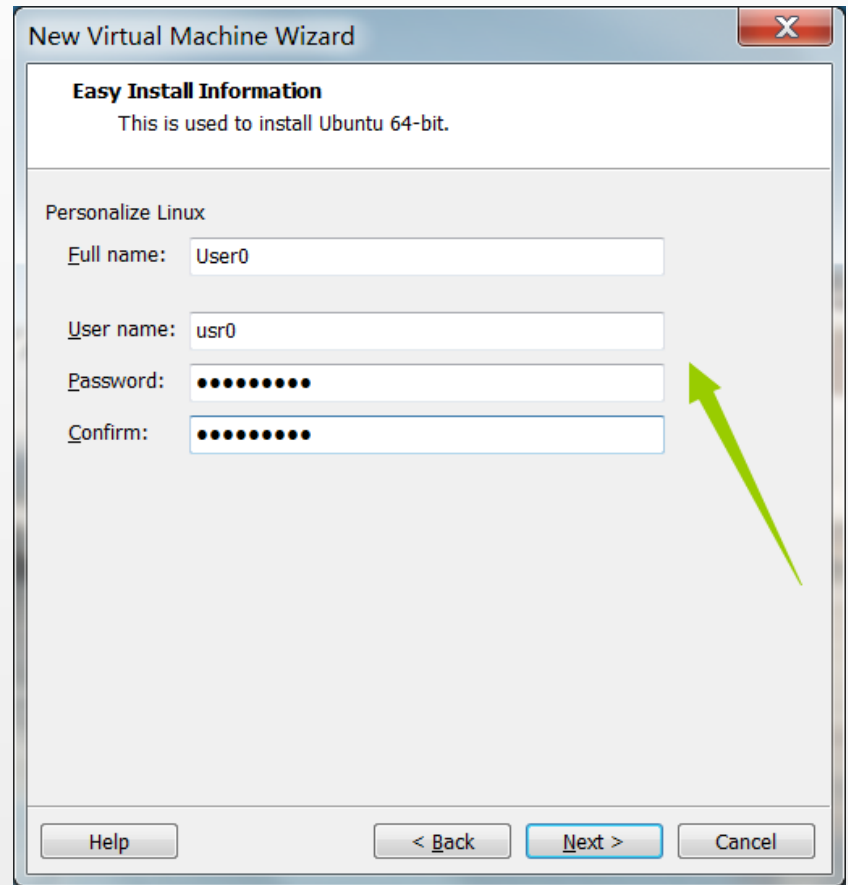


Install & Configure a Virtual

Create a Virtual Machine



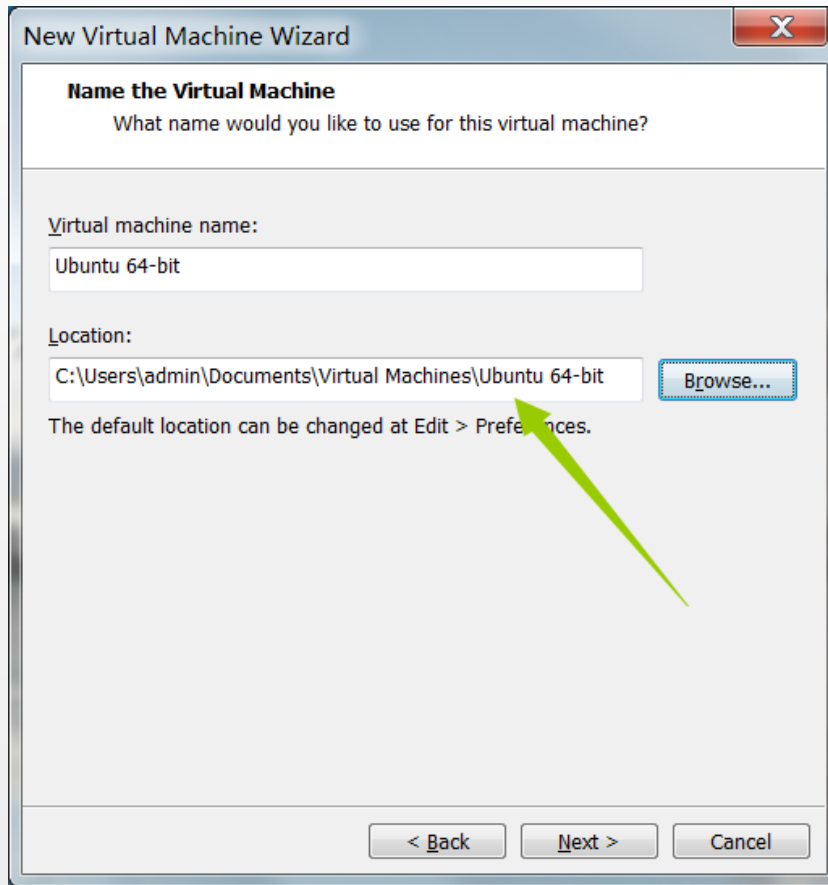
Select the .iso file of ubuntu
downloaded before



Fill the user name and
the password of the super user

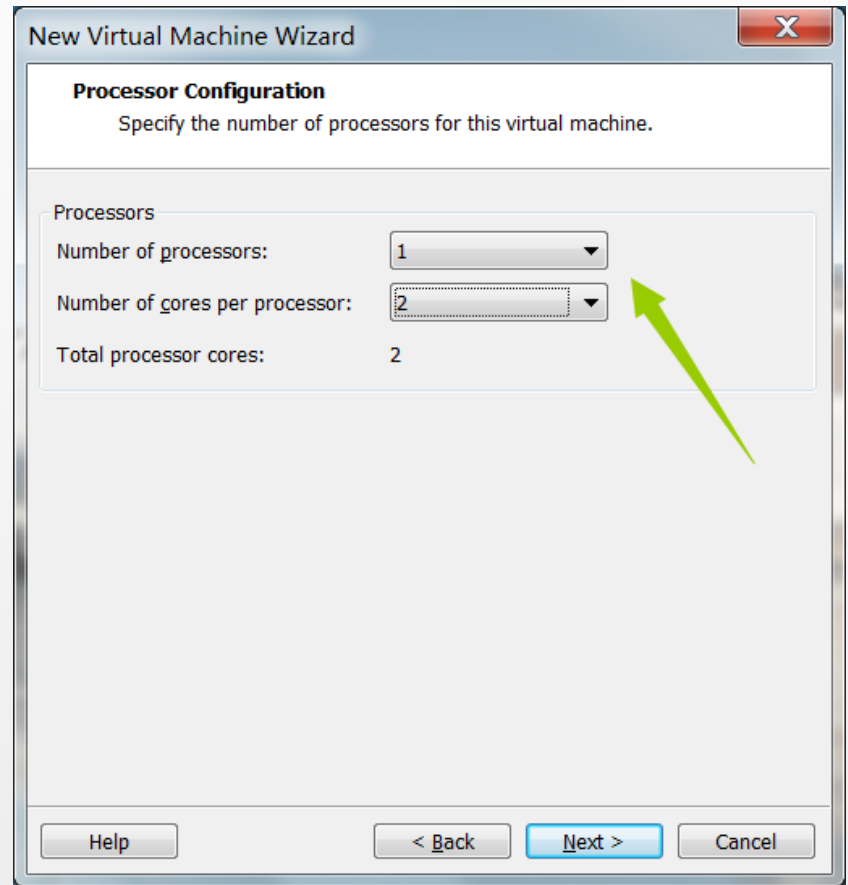
Install & Configure a Virtual

Create a Virtual Machine



The screenshot shows the 'Name the Virtual Machine' step of the 'New Virtual Machine Wizard'. The window title is 'New Virtual Machine Wizard'. The subtitle 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:' with the text 'Ubuntu 64-bit' and 'Location:' with the text '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 buttons for '< Back', 'Next >', and 'Cancel'.

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

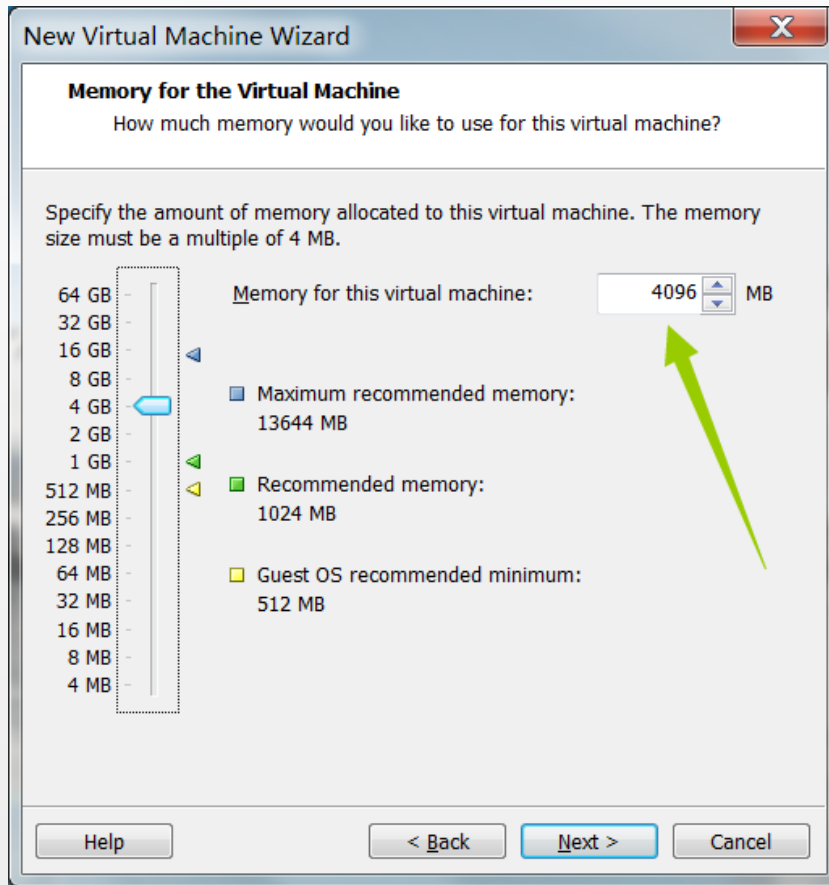


The screenshot shows the 'Processor Configuration' step of the 'New Virtual Machine Wizard'. The window title is 'New Virtual Machine Wizard'. The subtitle is 'Processor Configuration' with the instruction 'Specify the number of processors for this virtual machine.'. Under the 'Processors' section, there 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 buttons for 'Help', '< Back', 'Next >', and 'Cancel'.

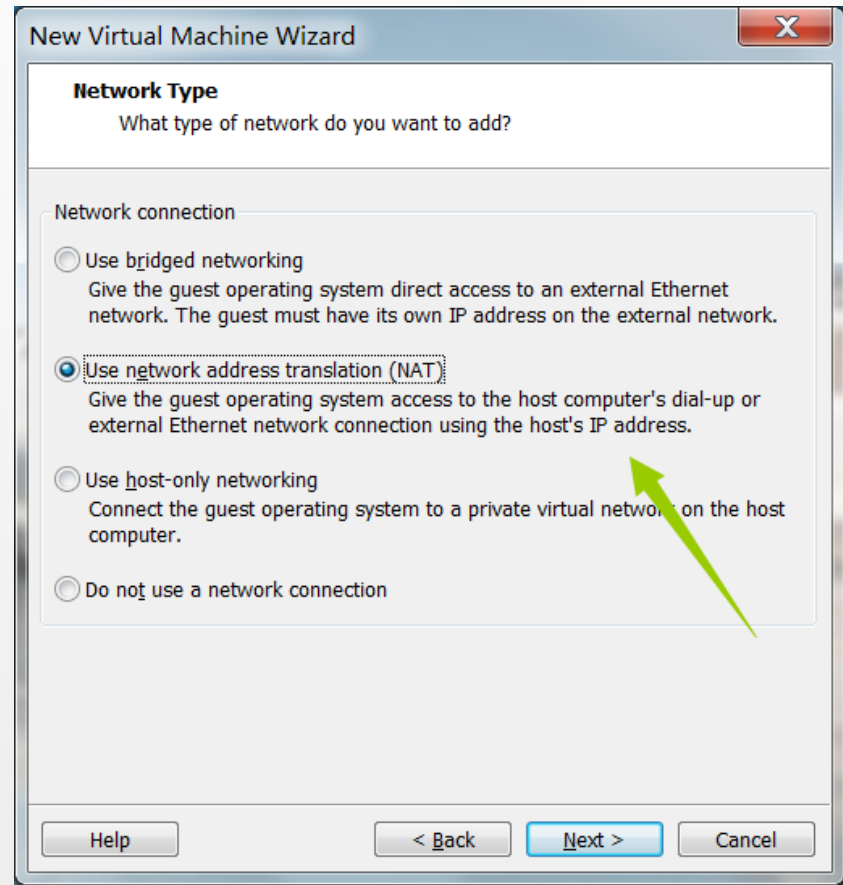
Set the number of processors for VM
1 processor 1 cores are enough

Install & Configure a Virtual

Create a Virtual Machine



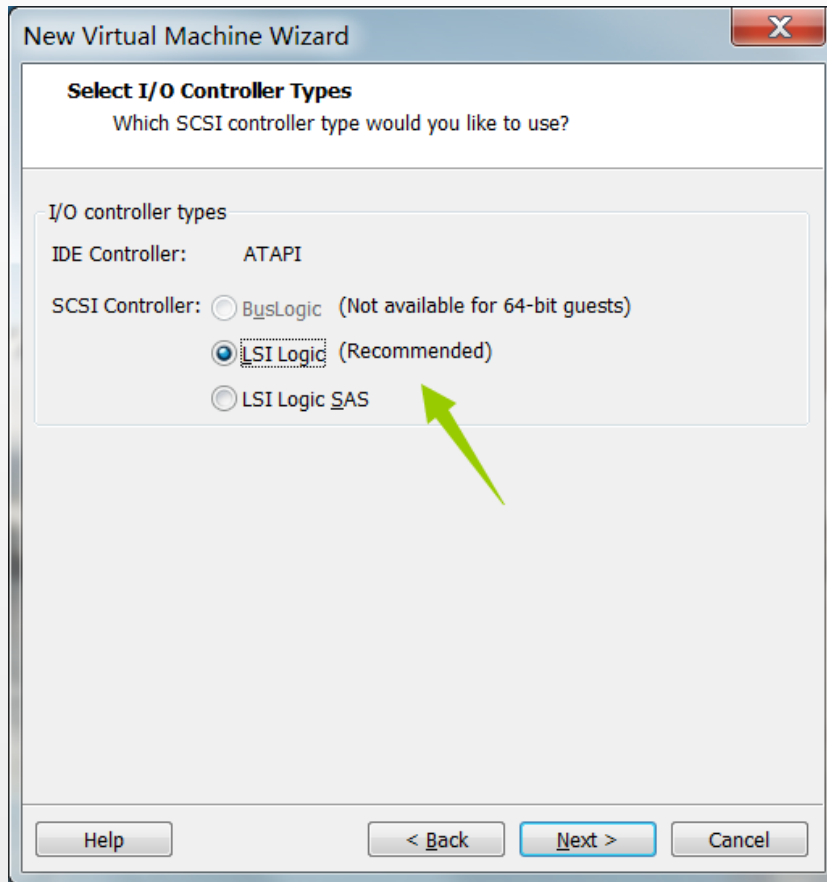
Set the memory for VM
more than 1024MB



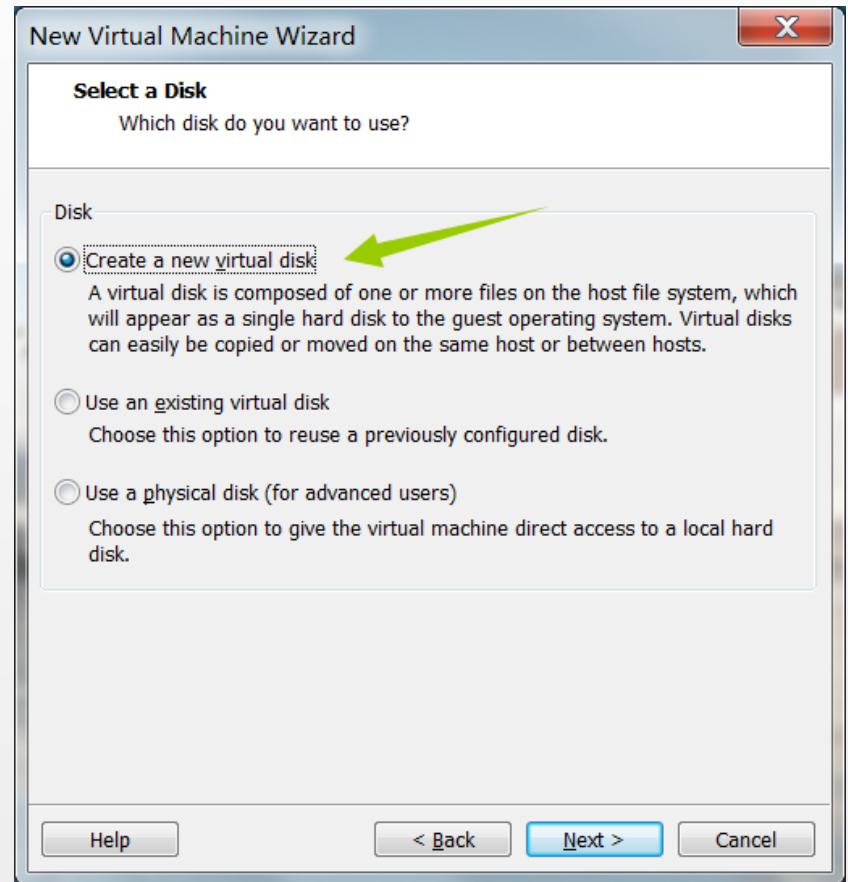
Set the network type
NAT

Install & Configure a Virtual

Create a Virtual Machine



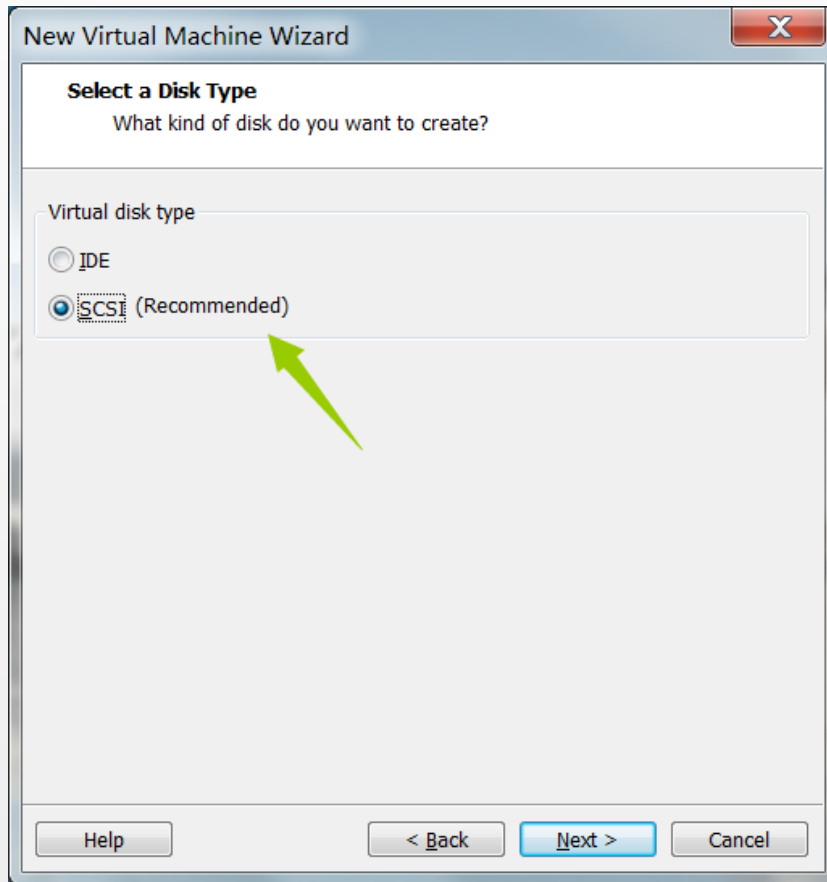
Set the I/O controller type
default



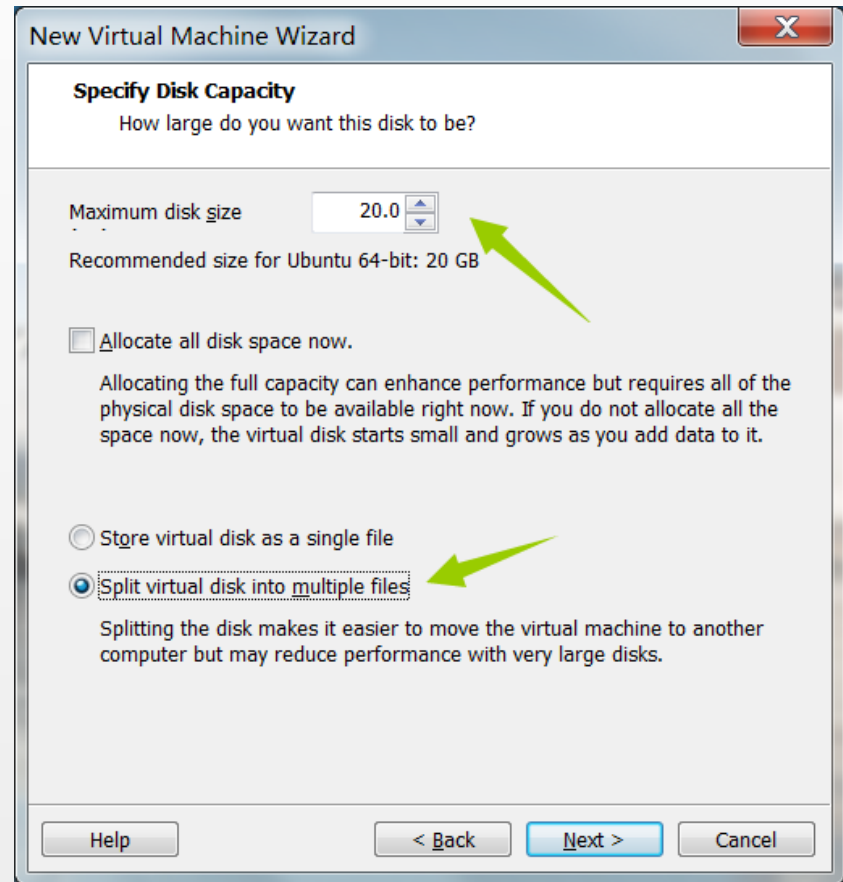
Create a virtual disk
composed of files on host OS

Install & Configure a Virtual

Create a Virtual Machine



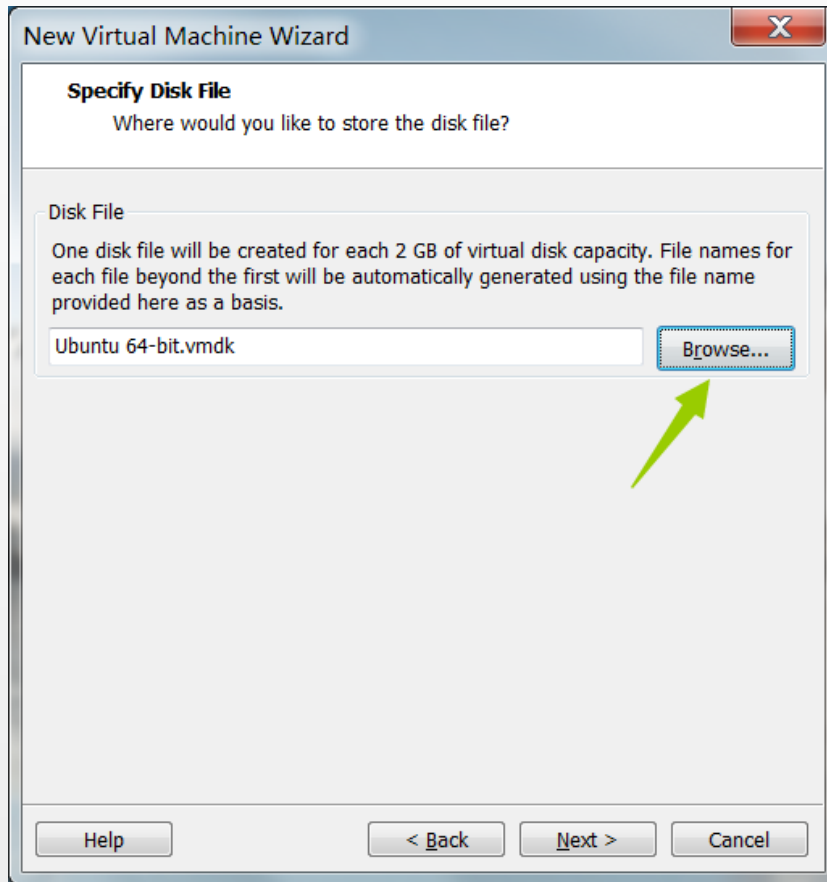
Set the disk type
default



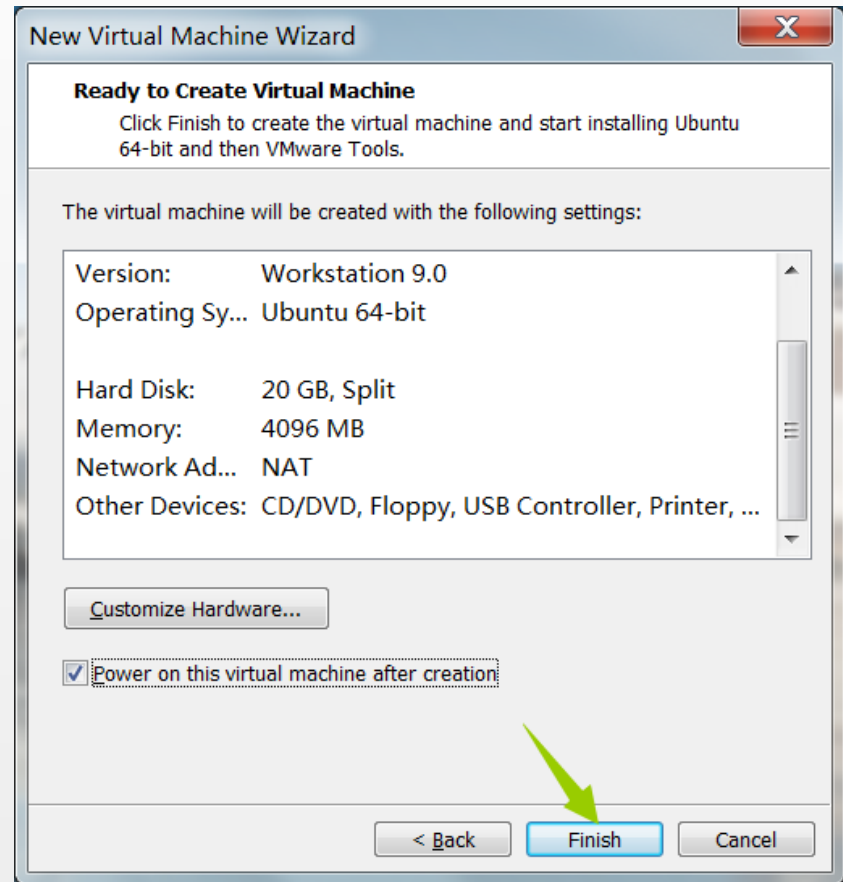
disk size: 15GB is enough
Split into multiple files: easy to move

Install & Configure a Virtual

Create a Virtual Machine



Select the location of disk files



Finish

PART I

- Brief Introduction
- Basic Conceptions & Environment
- Install & Configure a Virtual Machine
- **Basic Commands**

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 rmdir
- 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
```

lrwxr-xr-x	1	root	root	1021112	Oct	7	2014	bash
↓	↓	↓	↓	↓	↓		↓	
permission		own		size (byte)			file (or dir.) name	
	link number	group			creating date			
	(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 /home/test/dir1`

Create the directory named “dir1”, if the path “/home/test” exists.

`$ mkdir -p /home/test/dir1`

If the path “/home/test” doesn’t exist, create each directories in it.

rmdir (remove **empty** directory)

`$ rmdir /home/test/dir1`

Remove a single directory named “dir1”.

`$ rmdir -p /home/test/dir1`

If it’s empty. Also remove the directories in the path

Basic Commands

rm (remove)

-r (--recursive) recursively delete directories.

if a directory being deleted has subdirectories,
delete them too.

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
```

Change the destination of standard output

```
$ cat file1 file2 > file3
```

Concatenate file1 file2, and output into file3. If file3 exists, it will be overwritten.

```
$ cat file1 file2 >> file3
```

The output will not overwrite the destination, but **attaching** to the back.

Basic Commands

| (pipeline)

```
$ command1 | command2
```

command1 has standard output, and command2 has standard input.

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

sort all of files and directories in “/bin” & “/usr/bin”

Basic Commands

xargs

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

```
$ find /bin -name "a*" | xargs ls -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)

A special editor for modifying files automatically.

It has several commands, but most people only learn the substitute command: s, which changes all occurrences of the regular expression into a new value.

```
$ sed s/aa/kk/ <test >test2
```

```
$ sed s/aa/kk/g <test >test2
```

Reference: <http://www.grymoire.com/Unix/Sed.html#uh-31>

Basic Commands

awk (Aho, Weinberg & Kernighan)

It is an excellent filter and report writer.

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

```
awk [-F field-separator] 'commands' input_file
```

```
$ awk -F: '{print $1,$5}' test
```

PART II

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

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`

`${VariableName}`

<code>\$0</code>	filename of the script
<code>\$n</code>	the n-th argument
<code>\$#</code>	the number of the arguments
<code>\$HOME</code>	user directory
<code>\$\$</code>	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 $(2+3)**2$ 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}

```
string="alibaba is a great  
company"  
echo ${string:1:4}  
#output : liba
```

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

Branches

Example :

test.sh

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

```
YACCESS=`date -d yesterday +%Y%m%d`
```

```
FILE="access_${YACCESS}.log.tgz"
```

```
if [ -f "$FILE" ];then
```

```
echo "OK"
```

```
else
```

```
echo "error $FILE"
```

```
fi
```

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

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

Compile & Debug (for

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

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

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

Makefiles

build the program **automately** according to the **makefile**

Makefiles are based on rules as:

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

Compile 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

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	execute a line of source code
p <variable name>	print the value of a variable
q	quit
help <command>	

PART II

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

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`

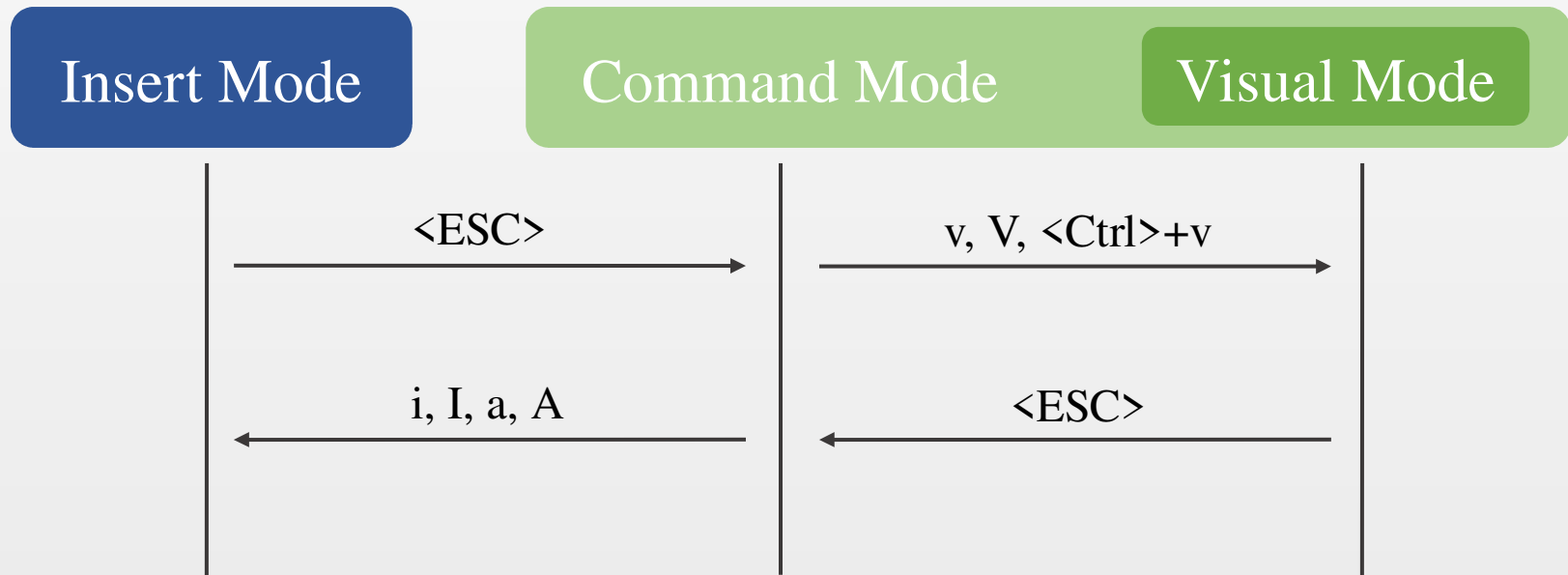
Text Editor (Vim)

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

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

Scroll the Screen

<Ctrl>+f	1 page
<Ctrl>+d	1/2 page
<Ctrl>+e	1 line
<Ctrl>+y	1 line
<Ctrl>+u	1/2 page
<Ctrl>+b	1 page



Text Editor (Vim)

Movement of the Cursor

j = ↑ **k** = ↓ **h** = ← **l** = →

0 first column of the line

^ first non-blank character of the line

w jump to next word

W jump to next word, ignore punctuation

e jump to word-end

E jump to word-end, ignore punctuation

b jump to word-beginning

B jump to word-beginning, ignore punctuation

ge jump to previous word-ending

gE jump to previous word-ending, ignore punctuation

g_ jump to last non-blank character of the line

\$ jump to the last character of the line

% 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

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

- Completion
- Searching & Replacing
- Marks