



LPIC-1

Fanavaran Anisa

Iran Linux House

Linux & Open Source Training Center



whoami

Mohsen Mohammad Amini

Email: m.mo.amini@gmail.com

Linked In: mohsen-Mohammadamini



cat /etc/passwd

let's all introduce ourselves to each other

(Name, Age, Education, Job, Reason/Goal)



LPIC-1 Certification

**LPIC-1
Exam 101**



**LPIC-1
Exam 102**



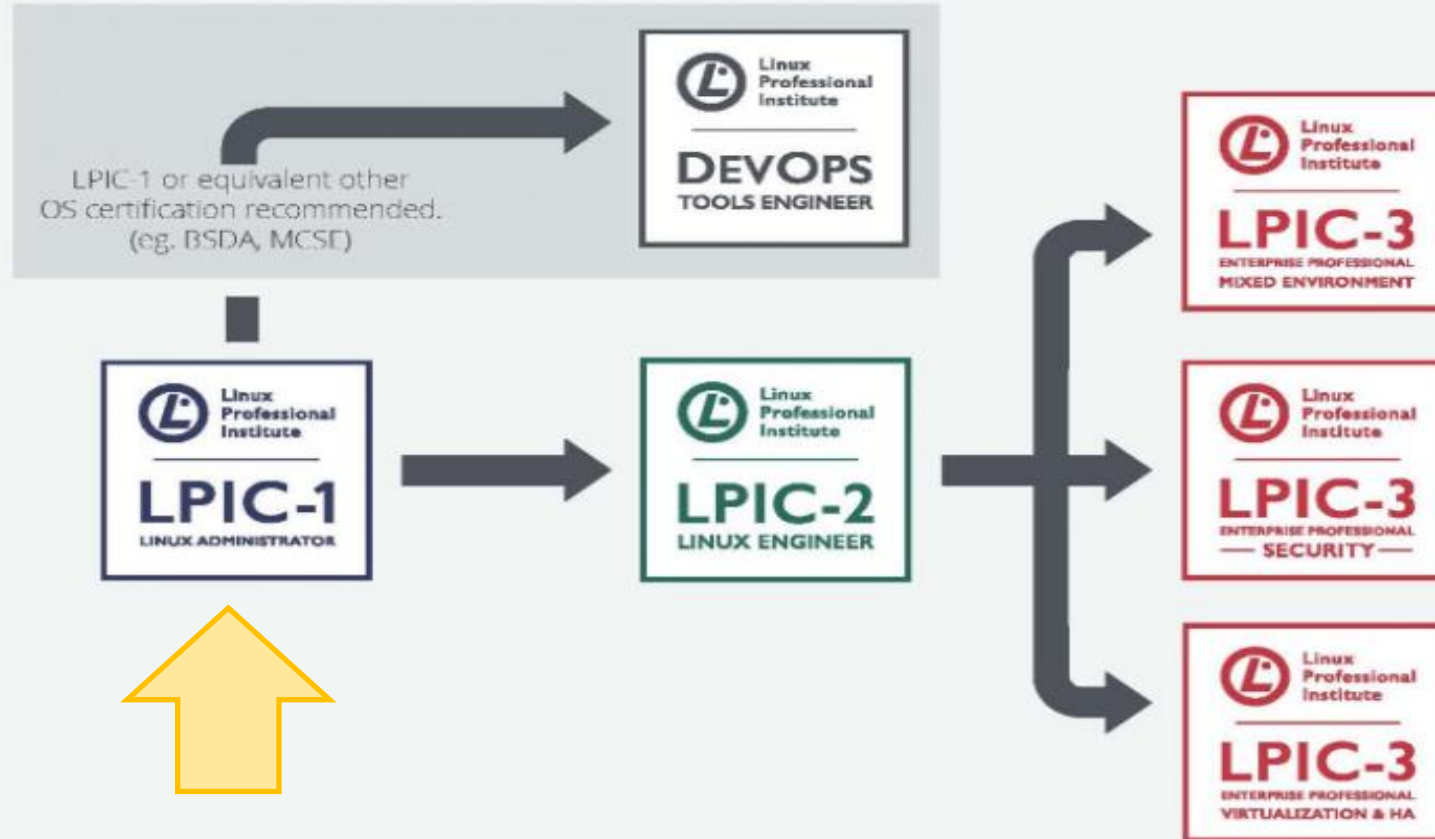
LPI Road Map

Educational Certificate



Build confidence

Professional Certification Track



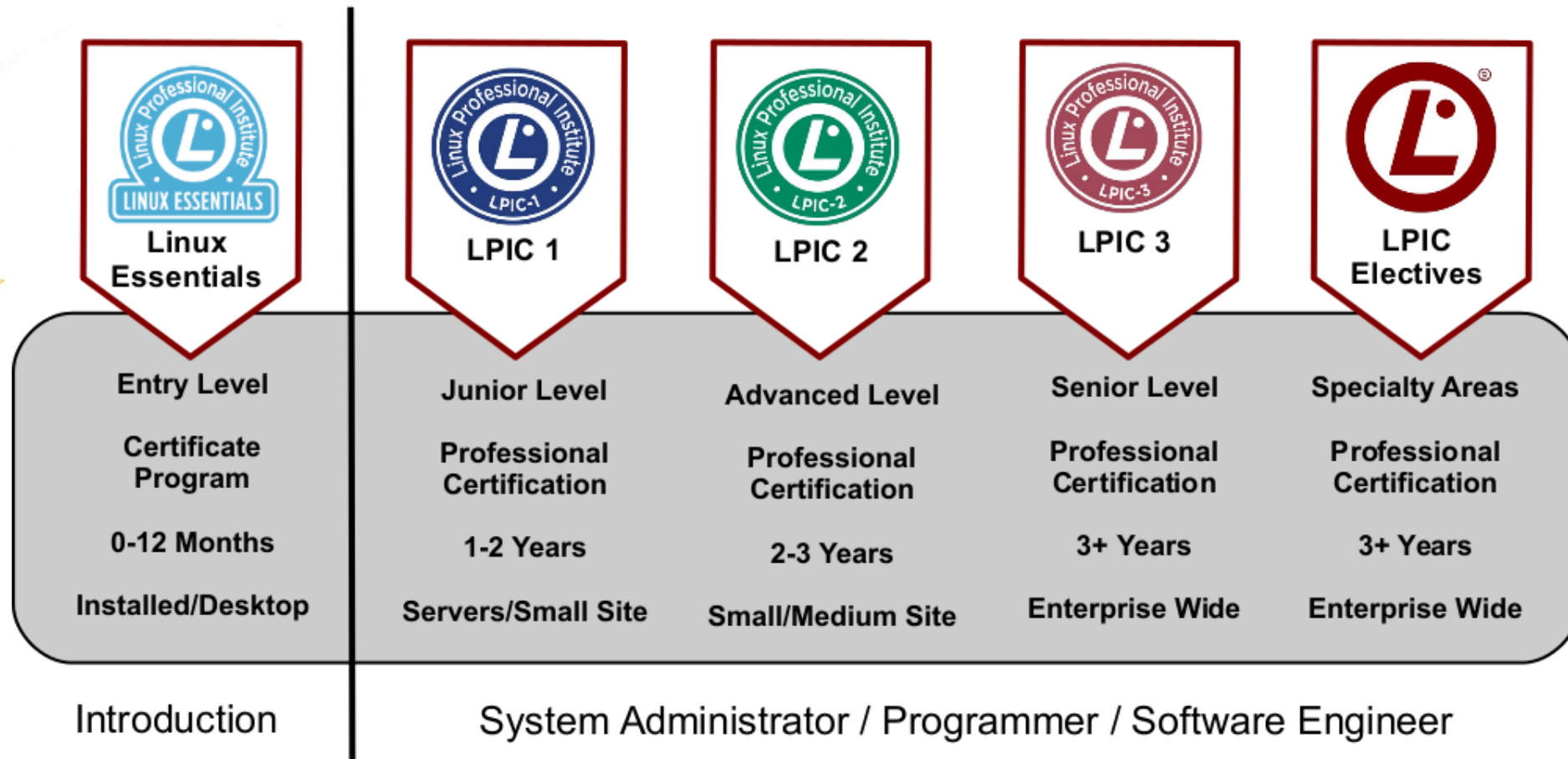
Unlock opportunities

Lead new projects

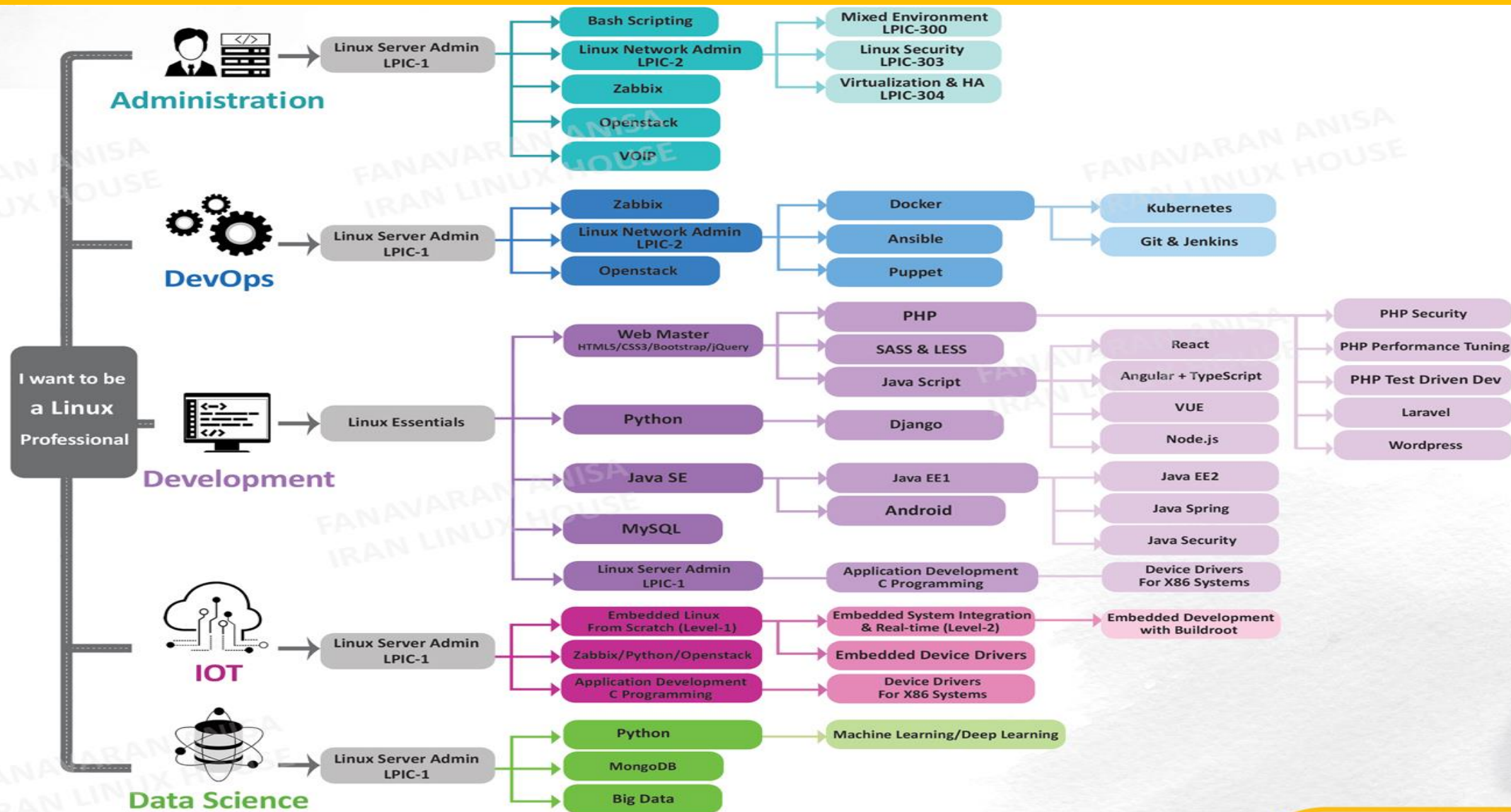
Achieve mastery

LPI Road Map

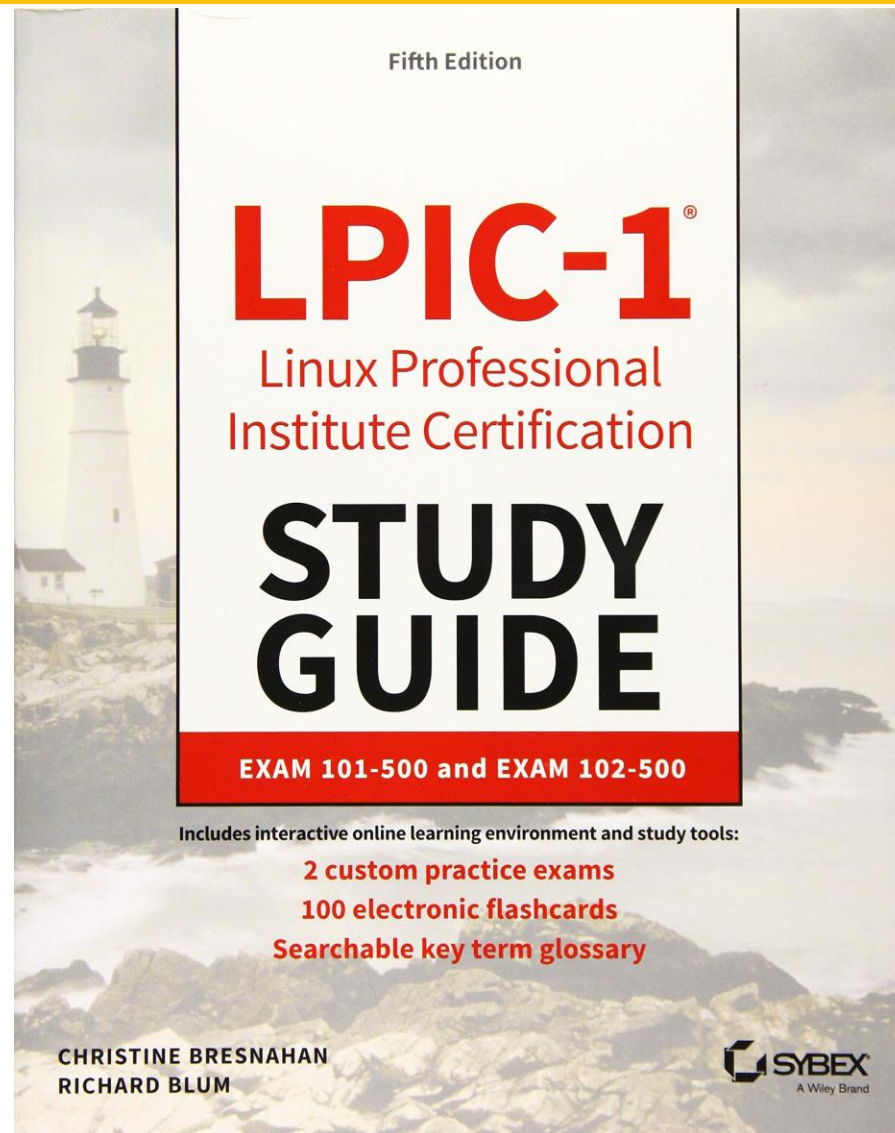
LPI Certification Schedule



Open-Source Training Roadmap



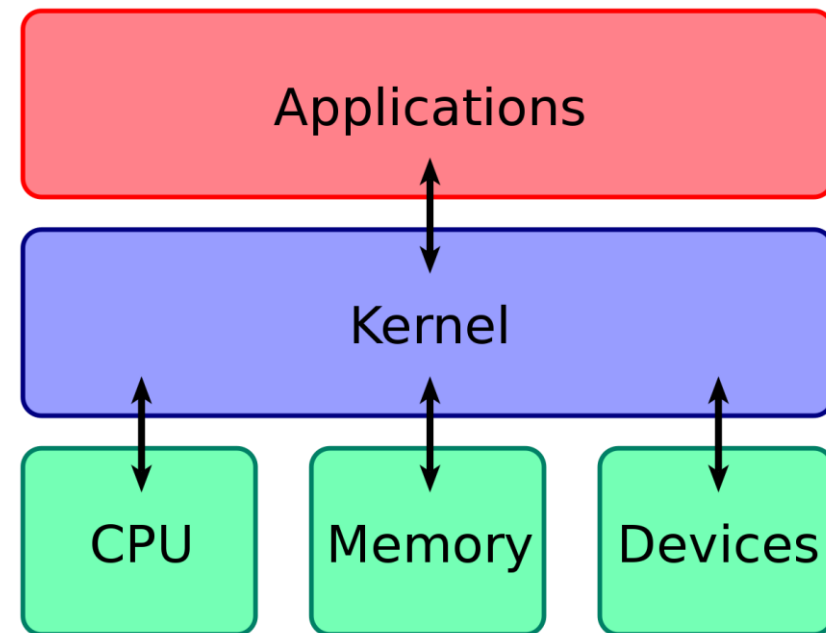
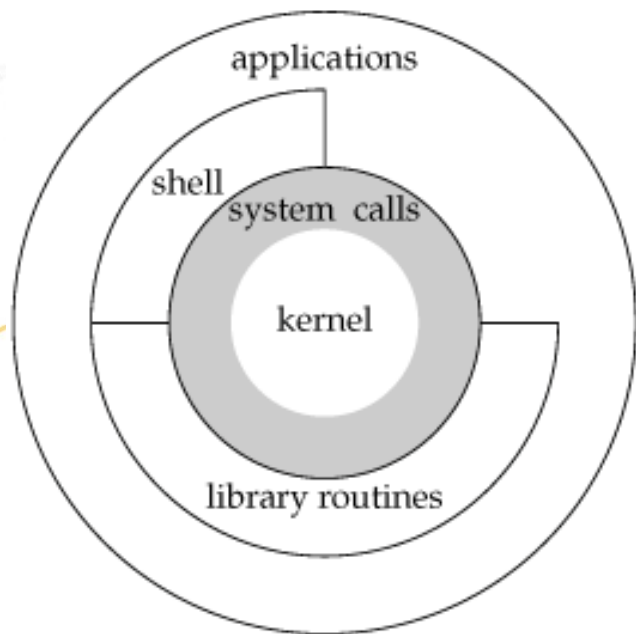
Reference Book



QUICK OVERVIEW



Operating System



Operating System

- ❖ An operating system, or OS, provides all of the most fundamental features of a computer, at least from a software point of view.
- ❖ An OS enables you to use the computer's hardware devices, defines the user interface standards, and provides the basic tools that begin to make the computer useful.
- ❖ Many of these features trace their way back to the OS's kernel.

Kernel

An OS kernel is a software component that is responsible for managing various low-level features of a computer, including:

- ✓ Interfacing with hardware devices(hard disk, network adapter and etc)
- ✓ Allocating memory to individual programs
- ✓ Allocating CPU time to individual programs
- ✓ Enabling programs to interact with each other

What is Linux?



What is Linux?

- ❖ Linux is a clone of Unix OS that has been popular in academic and business environments for years.
- ❖ Linux consists of a kernel, which is the core control software and many libraries and utilities that rely on the kernel to provide features with which users interact.

Discussing Distributions



- ❖ Think of the Linux kernel as a car's engine and a distribution as the car's features.
- ❖ Between manufacturers and models, car features are often different.
- ❖ This is also true with different distributions.

Distributions for LPIC-1

ubuntu 

Ubuntu Desktop 18-04 LTS



CentOS

CentOS 7

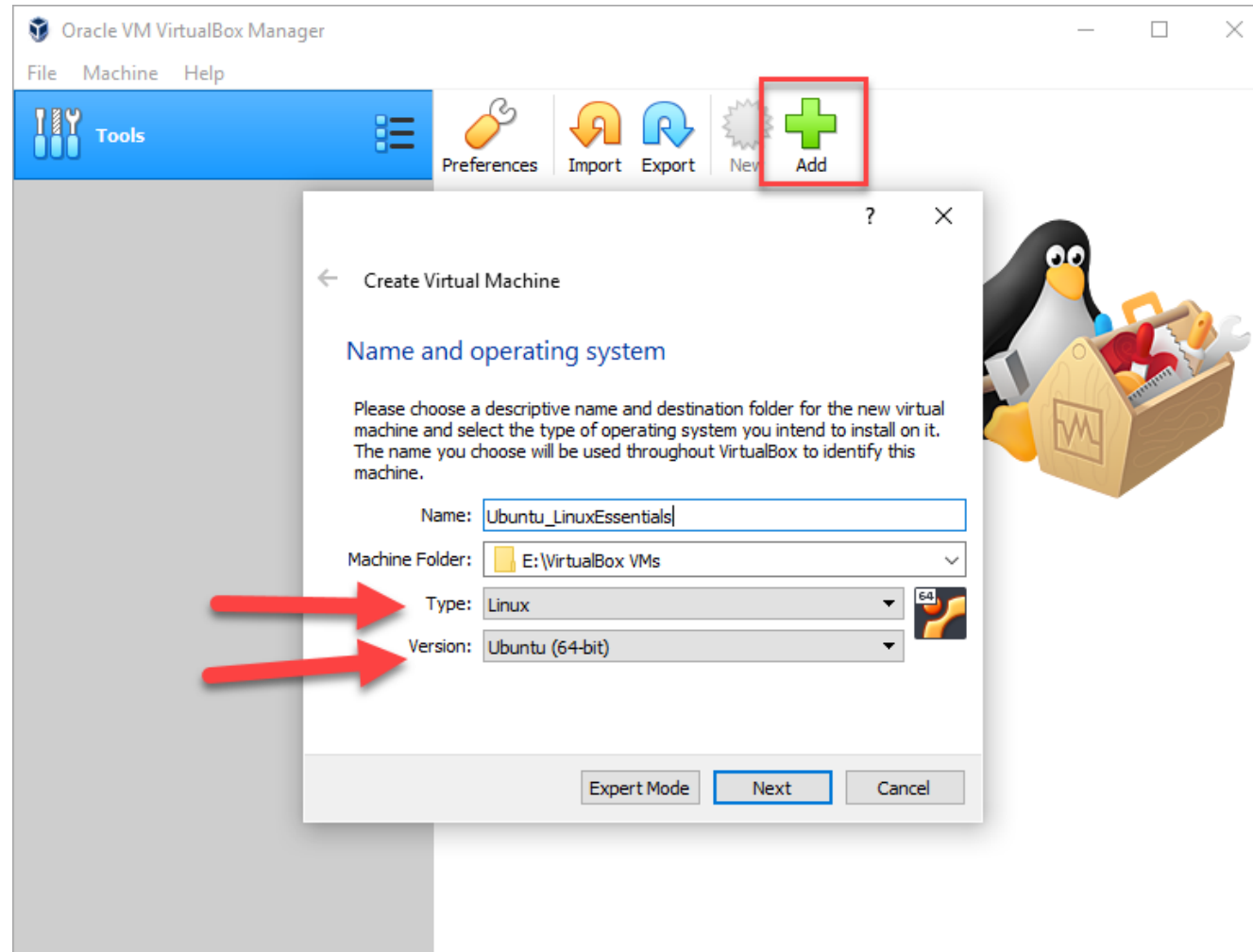
LINUX INSTALLATION

Ubuntu



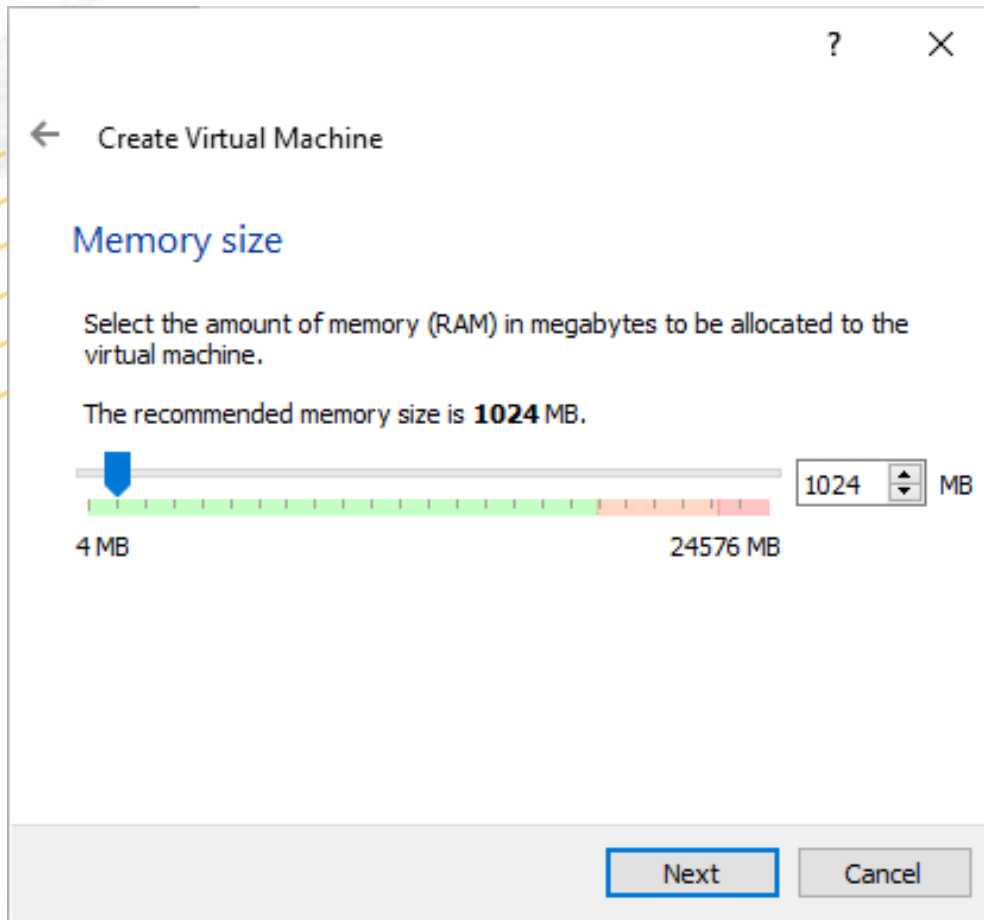
First Steps with Linux -Installation

1. Choosing the Operating System Category



First Steps with Linux -Installation

2. Choosing the OS's Memory Size



The screenshot shows the 'Create Virtual Machine' dialog box with the 'Memory size' tab selected. It instructs the user to select the amount of memory (RAM) in megabytes to be allocated to the virtual machine. The recommended memory size is 1024 MB. A slider bar is shown with a blue arrow pointing to 1024 MB. The slider ranges from 4 MB to 24576 MB. The 'Next' button is highlighted.

← Create Virtual Machine

Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

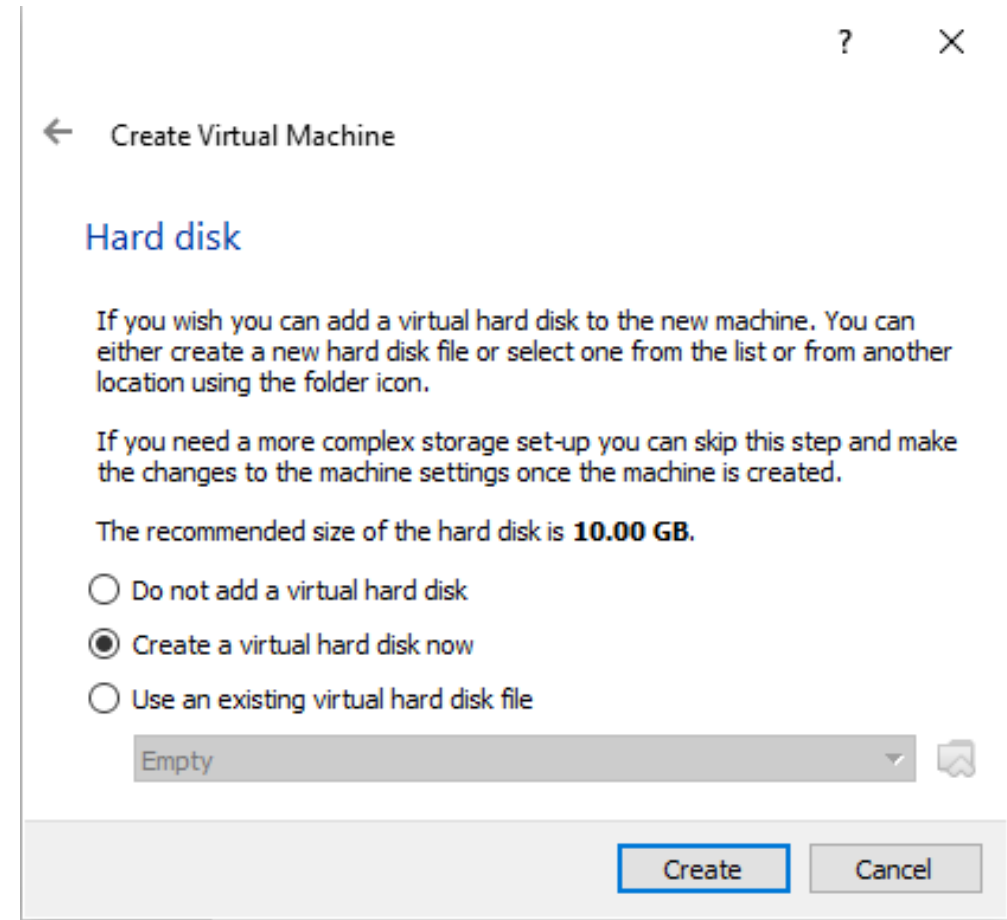
The recommended memory size is **1024 MB**.

4 MB 24576 MB

1024 MB

Next Cancel

3. Creating Virtual Hard Disk



The screenshot shows the 'Create Virtual Machine' dialog box with the 'Hard disk' tab selected. It instructs the user that if they wish, they can add a virtual hard disk to the new machine. They can either create a new hard disk file or select one from the list or from another location using the folder icon. It also states that if they need a more complex storage set-up, they can skip this step and make the changes to the machine settings once the machine is created. The recommended size of the hard disk is 10.00 GB. Three radio buttons are present: 'Do not add a virtual hard disk', 'Create a virtual hard disk now' (which is selected), and 'Use an existing virtual hard disk file'. Below the radio buttons is a text box containing 'Empty' and a folder icon. The 'Create' button is highlighted.

← Create Virtual Machine

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **10.00 GB**.

☐ Do not add a virtual hard disk

☒ Create a virtual hard disk now

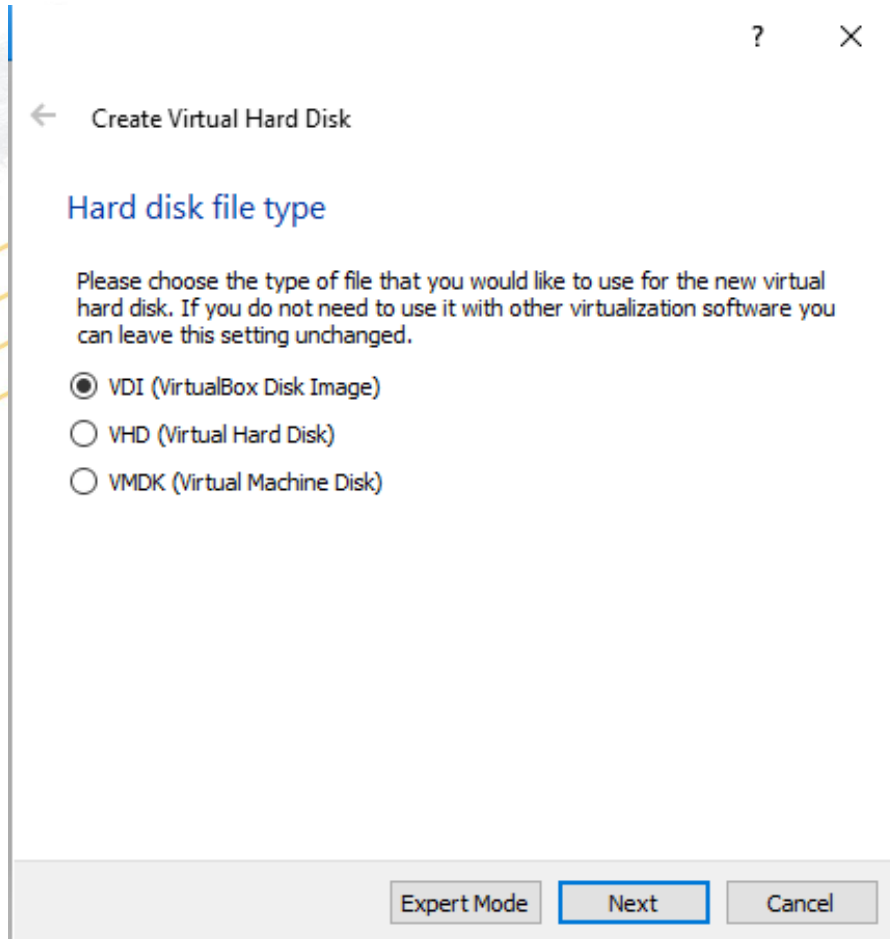
☐ Use an existing virtual hard disk file

Empty

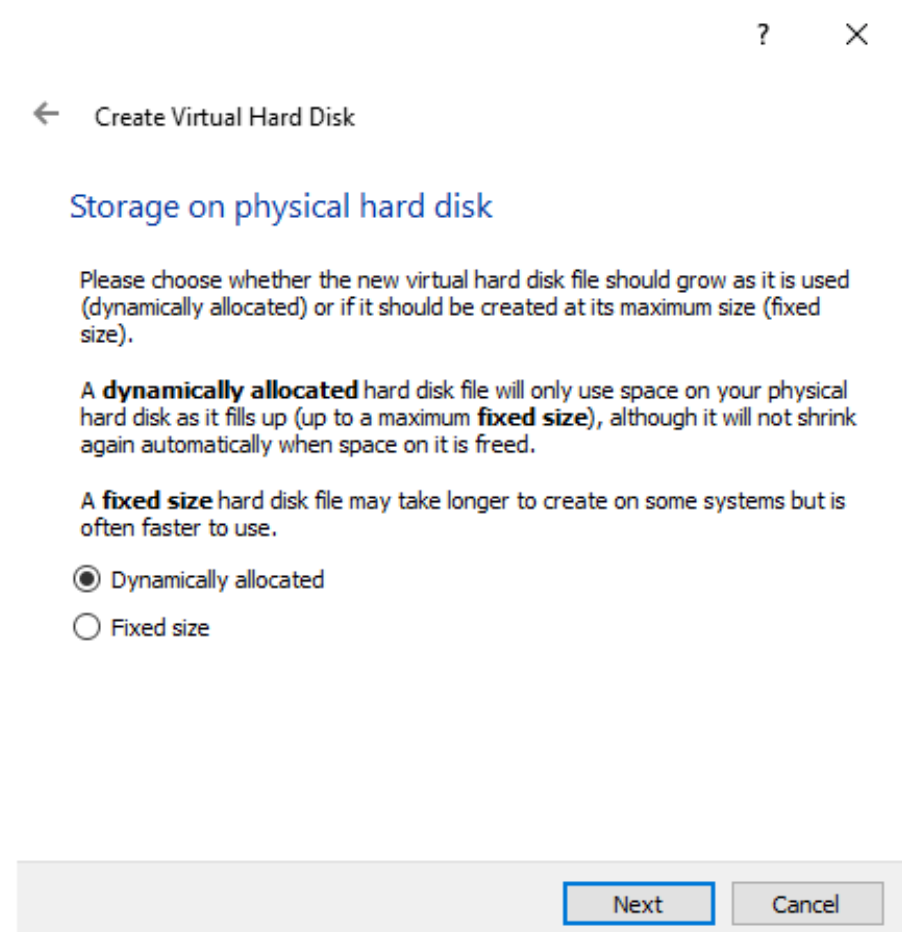
Create Cancel

First Steps with Linux -Installation

4. Choosing Hard Disk Type

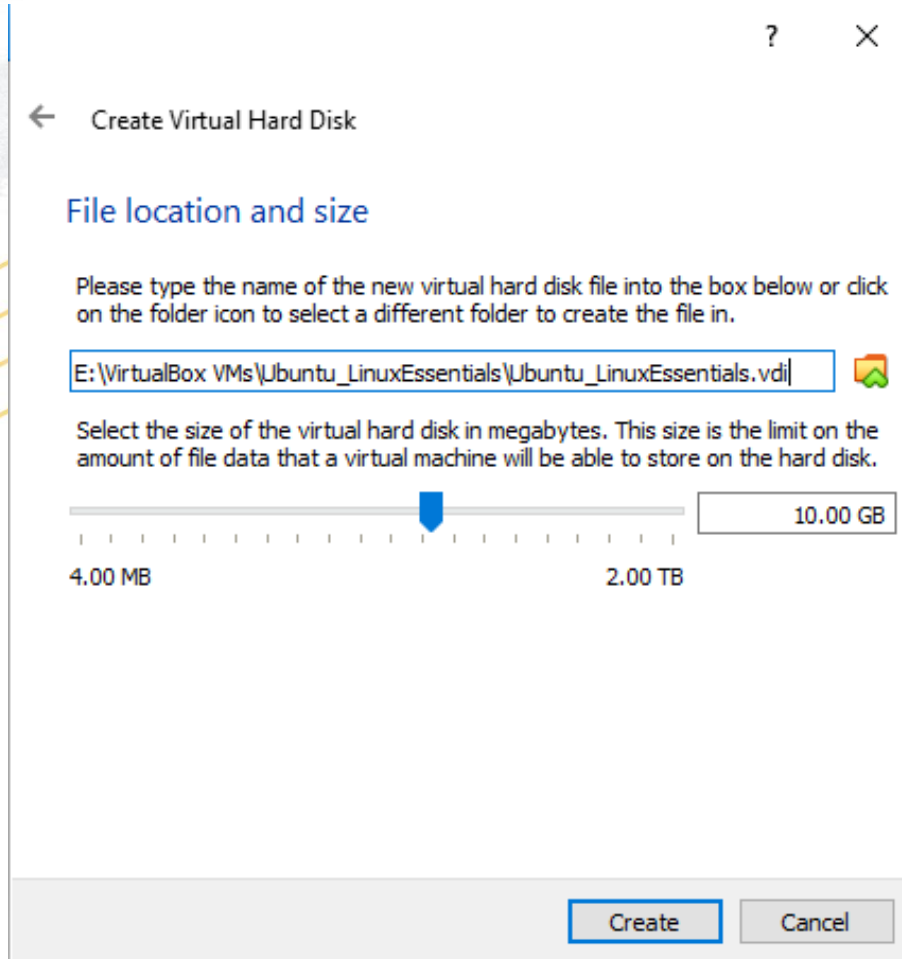


5. Choosing Hard Disk Growth Type

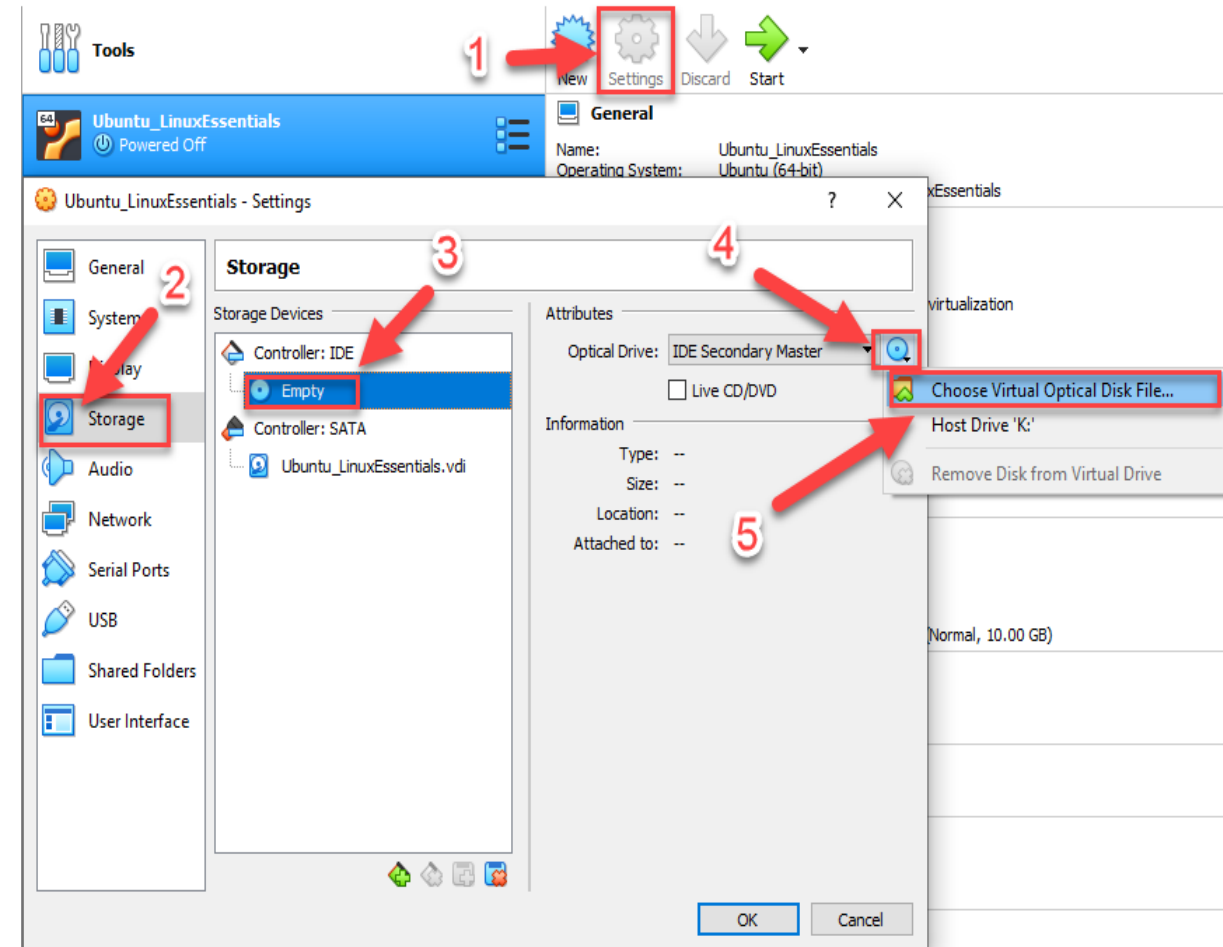


First Steps with Linux -Installation

6. Choosing File Location and Size

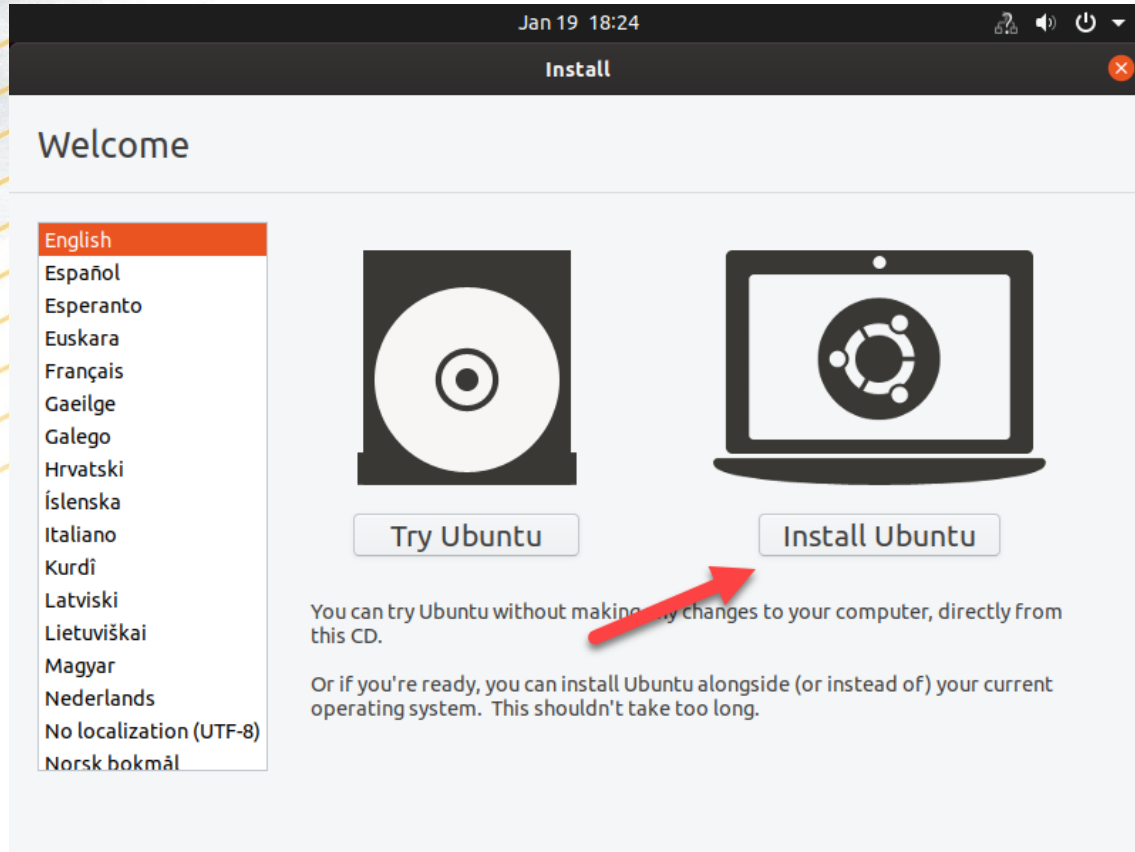


7. Choosing ISO File

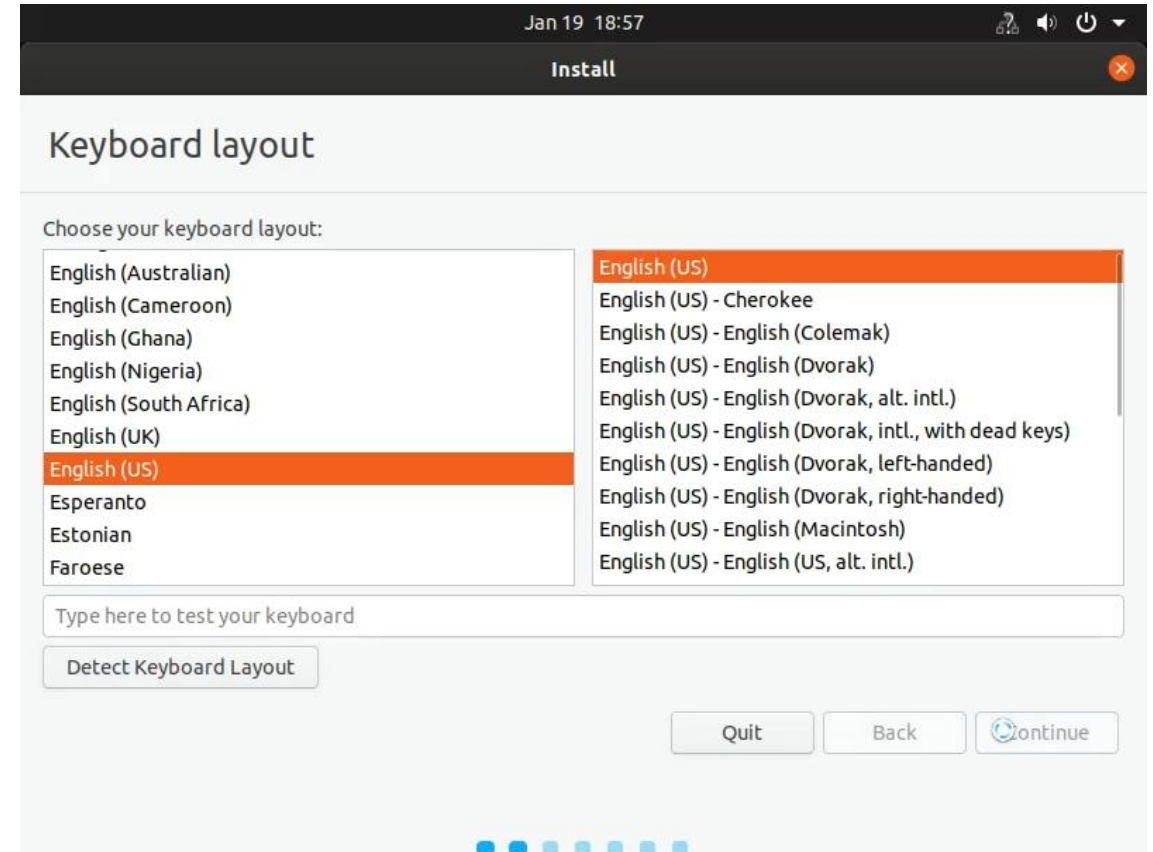


First Steps with Linux -Installation

8. Preparation

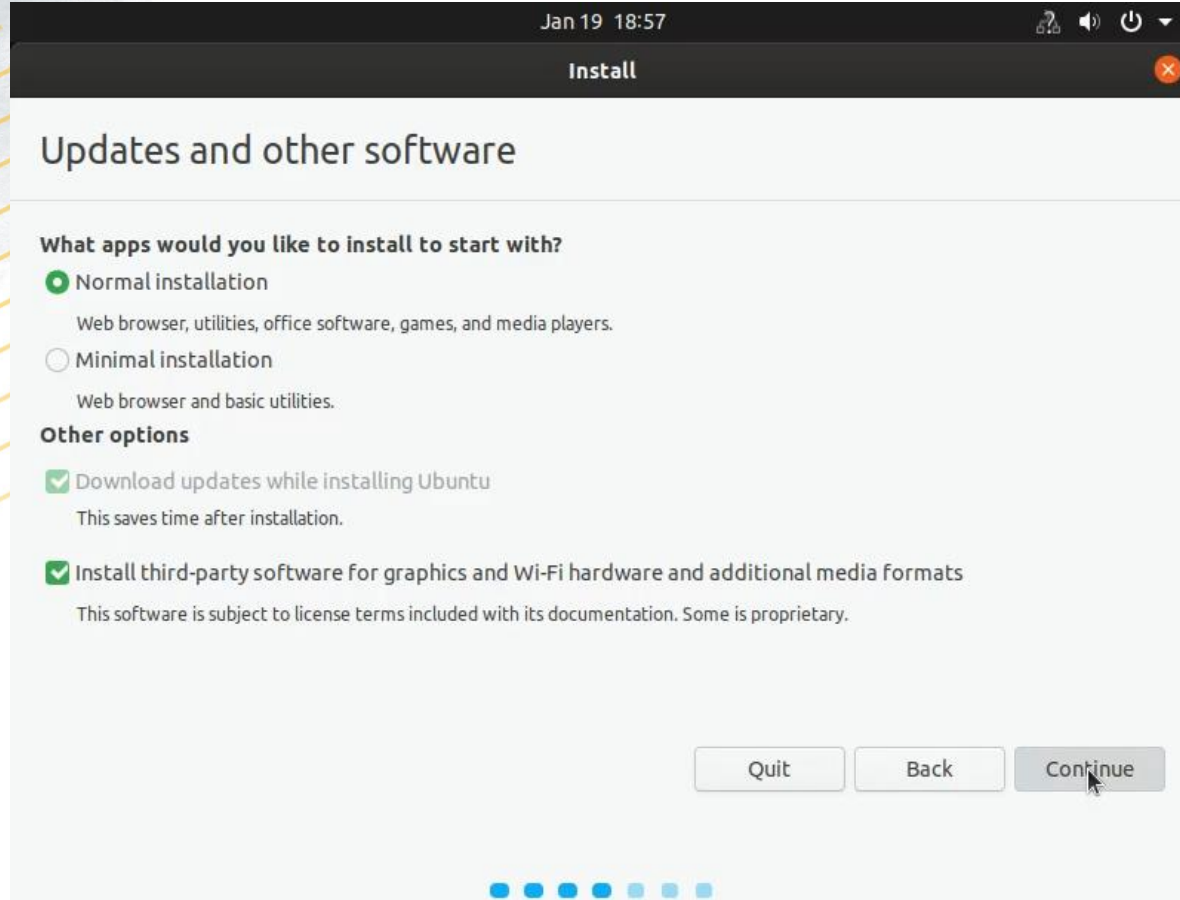


9. Keyboard Layout

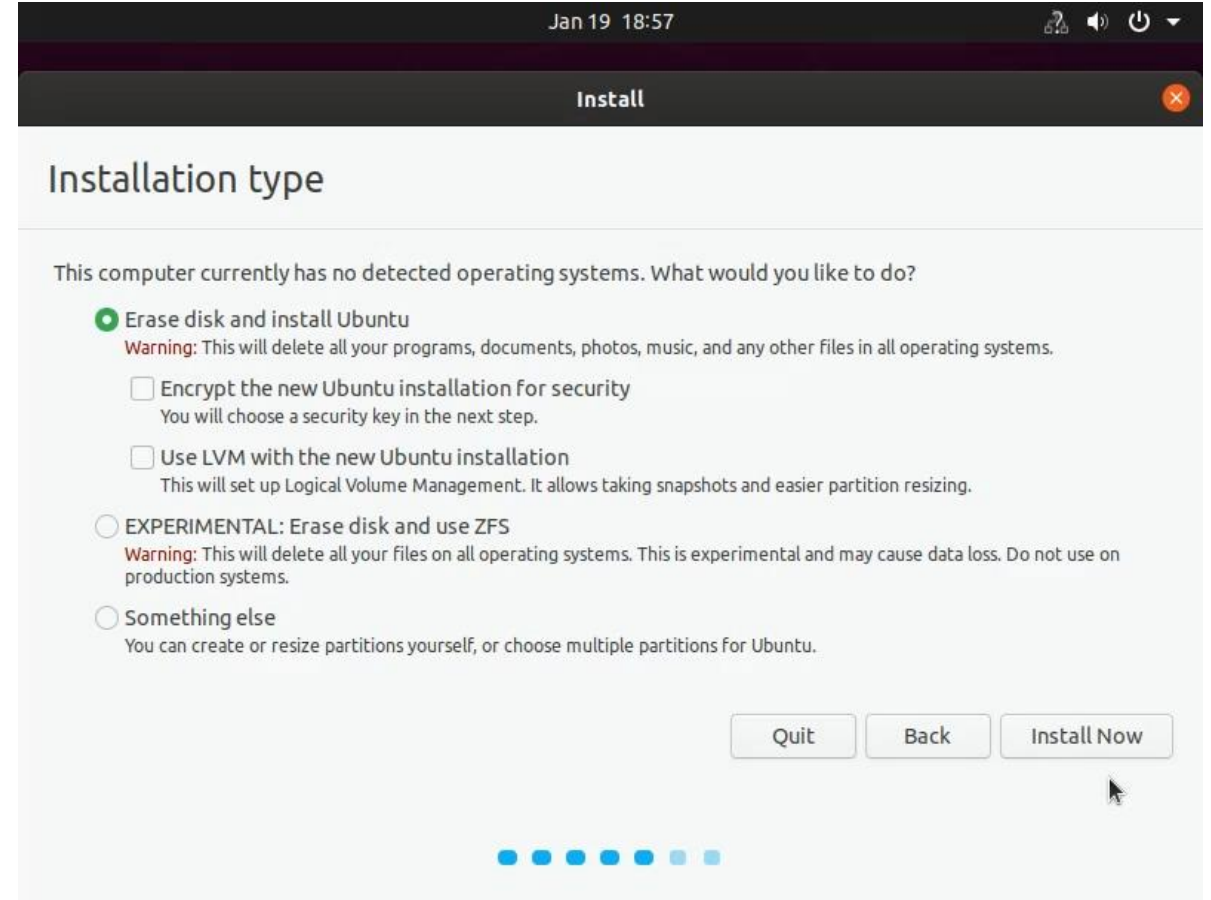


First Steps with Linux -Installation

10. Additional Software Installation



11. Installation Type




First Steps with Linux -Installation

12.Choosing Location

Jan 19 18:58

Install

Where are you?



Tehran

Back Continue

Progress indicator: 10 dots, 9th dot is highlighted.

13.Choosing Username and Password

Jan 19 22:39

Install

Who are you?

Your name: Anisa Essentials ✓

Your computer's name: anisa-VirtualBox ✓
The name it uses when it talks to other computers.

Pick a username: anisa ✓

Choose a password: ●●●●●● Fair password

Confirm your password: ●●●●●● ✓

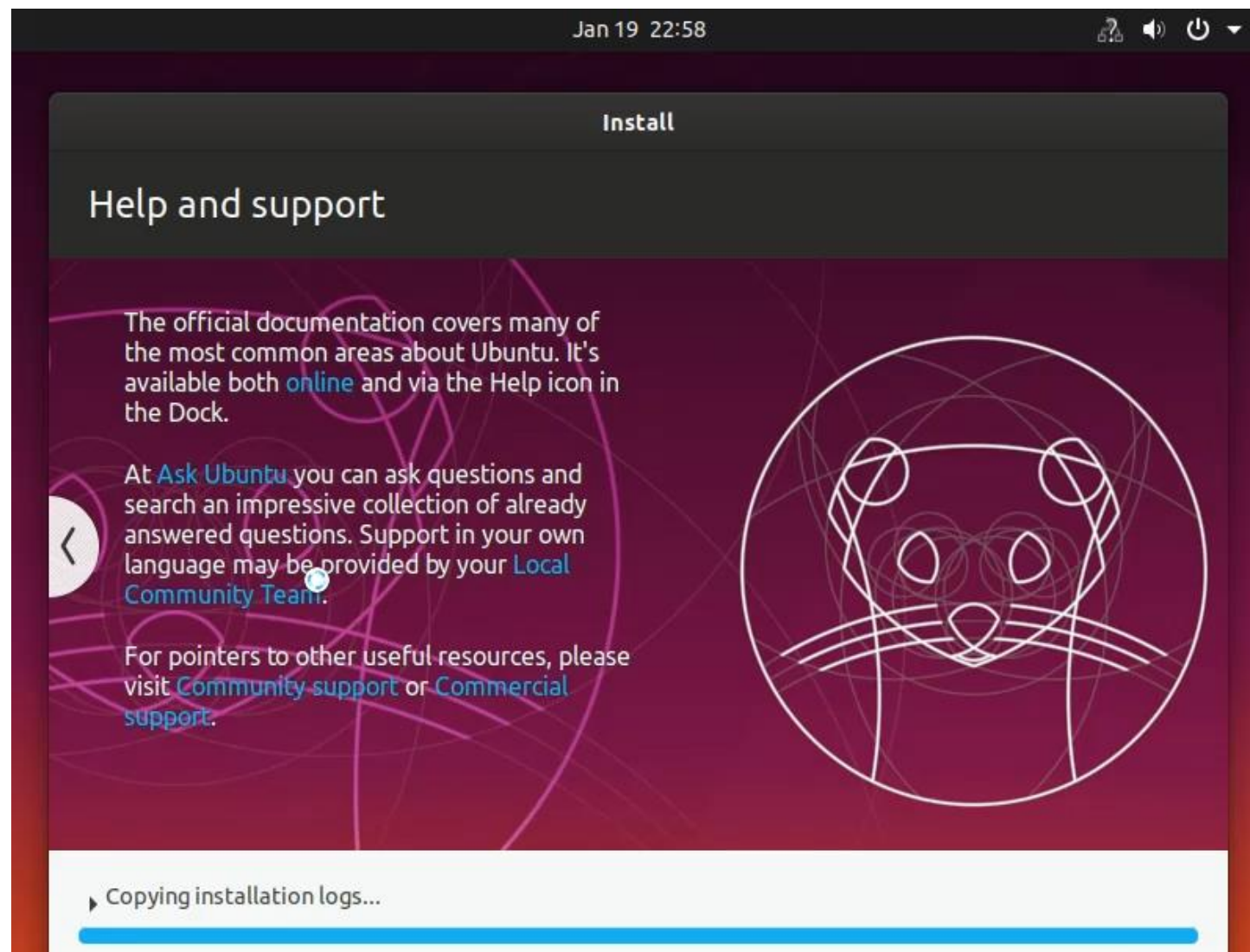
☐ Log in automatically

☒ Require my password to log in

Back Continue

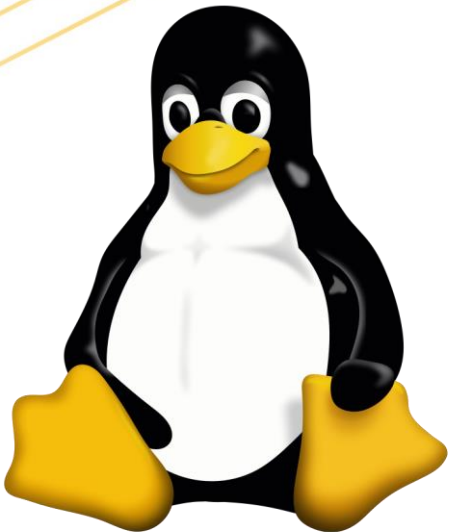
First Steps with Linux -Installation

14. Installing...



First Steps with Linux -Installation

15. Welcome to the GNU/LINUX



**KEEP
CALM
AND
ENJOY
LINUX**

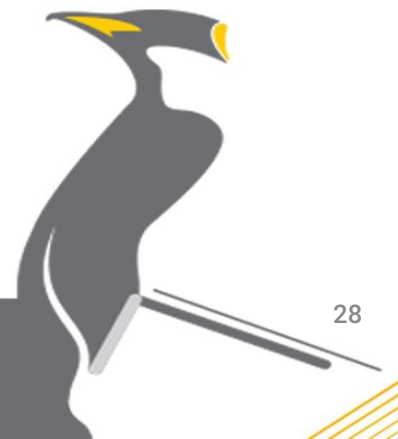
Exploring Linux Command-Line Tools

Objectives:

- ✓ 103.1 Work on the command line
- ✓ 103.2 Process text streams using filters
- ✓ 103.4 Use streams, pipes, and redirects
- ✓ 103.7 Search text files using regular expressions
- ✓ 103.8 Basic file editing



UNDERSTANDING COMMAND-LINE BASICS



System's Kernel

❖ **uname** - print system information

uname [OPTION]...

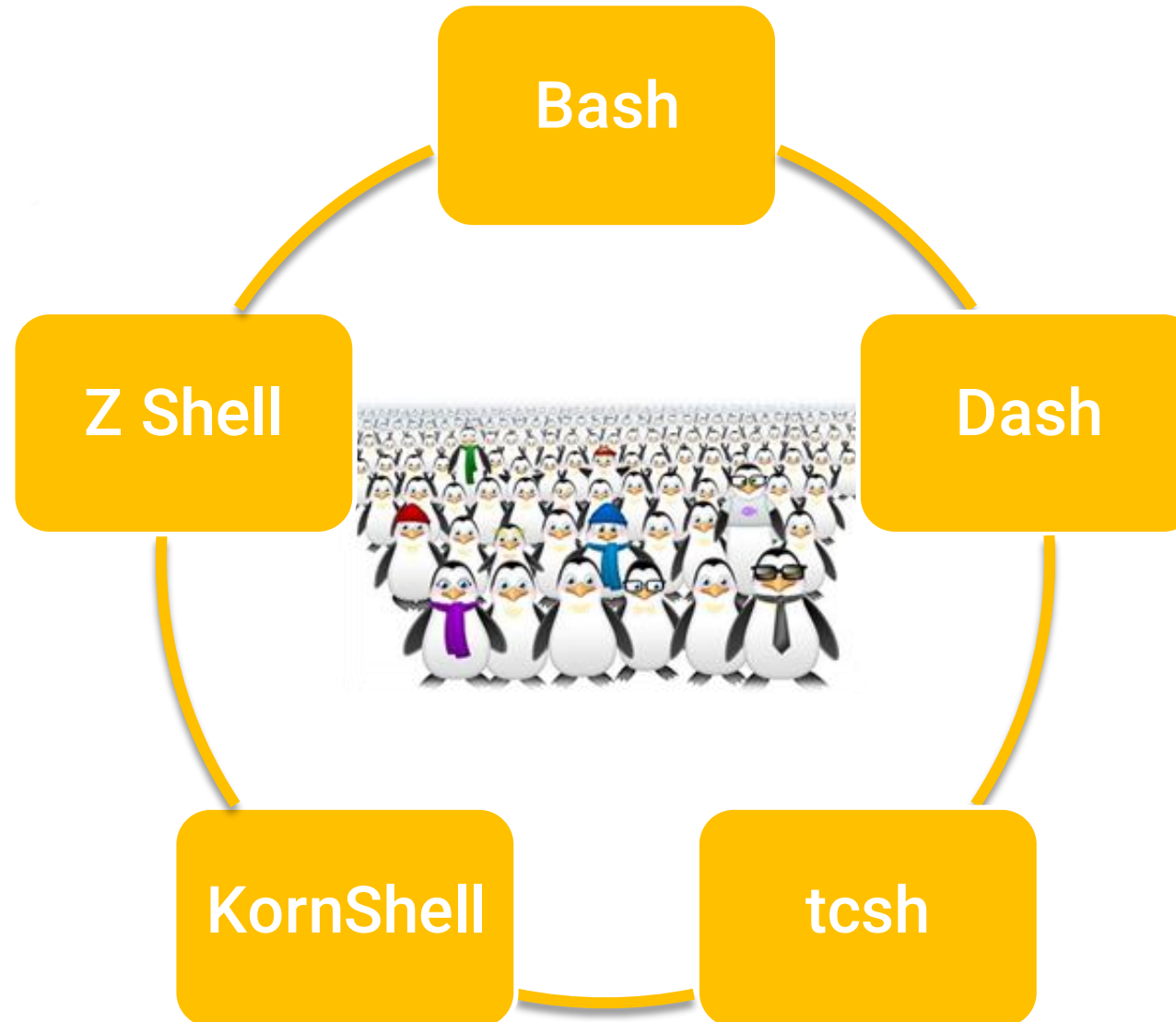
✓ **-r, --kernel-release**

- print the kernel release

✓ **-a, --all**

- print all information (kernel name, hostname, kernel release, kernel version, machine hardware name, processor type, hardware platform, operating system)

Most common shells



System's Shell

- ❖ When looking at shells, it is important to understand the history and current use of the `/bin/sh` file.
- ❖ Originally, this file was the location of the system's shell.
- ❖ Showing to which shell `/bin/sh` points

```
$ readlink /bin/sh
```

```
$ echo $SHELL
```

```
$ echo $BASH_VERSION
```

Using a Shell

❖ **echo** - display a line of text

echo [OPTION]... [STRING]...

✓ **-n** do not output the trailing newline

✓ **-e** enable interpretation of backslash escapes

\$ echo

\$ echo I Love Linux

Sequence	Interpreted as
<code>\\</code>	A literal backslash character ("").
<code>\a</code>	An alert (The BELL character).
<code>\b</code>	Backspace.
<code>\c</code>	Produce no further output after this.
<code>\e</code>	The escape character; equivalent to pressing the escape key.
<code>\f</code>	A form feed .
<code>\n</code>	A newline.
<code>\r</code>	A carriage return .
<code>\t</code>	A horizontal tab.
<code>\v</code>	A vertical tab.



Quoting Metacharacters

- ❖ Within the Bash shell are several characters that have special meanings and functions.
- ❖ These characters are called metacharacters. Bash shell metacharacters include the following:

✓ * ? [] ' " \ \$; & () | ^ < >

Quoting Metacharacters

```
$ echo $SHELL  
/bin/bash
```

```
$ echo It cost $1.00  
It cost .00
```

```
$ echo It cost \$1.00  
It cost $1.00
```

```
$ echo Is Schrodinger\'s cat alive or dead\  
Is Schrodinger's cat alive or dead?
```

```
$ echo Is "Schrodinger's" cat alive or "dead?"  
Is Schrodinger's cat alive or dead?
```

```
$ echo "Is Schrodinger's cat alive or dead?"  
Is Schrodinger's cat alive or dead?
```

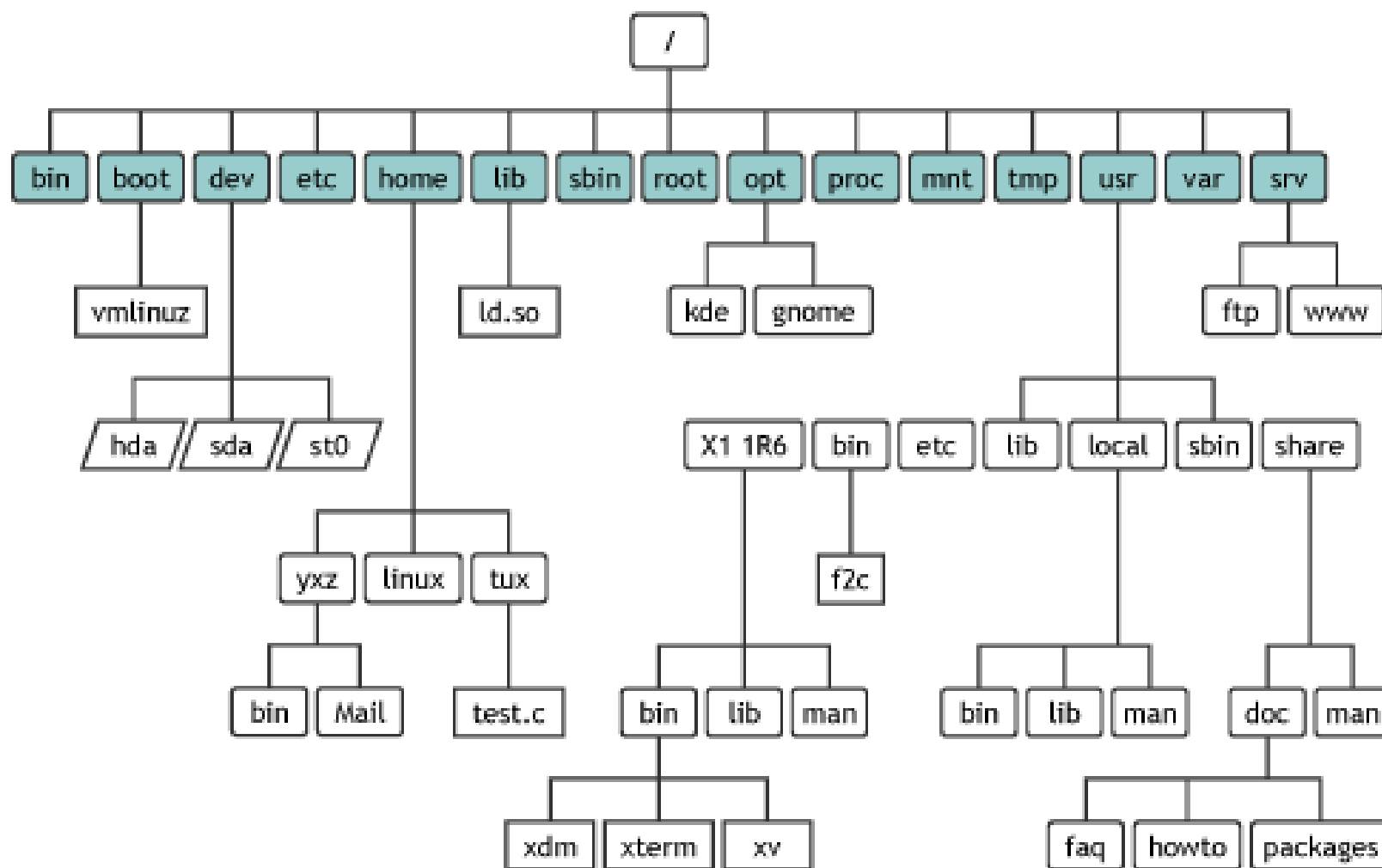
```
$ echo 'Is Schrodinger's cat alive or dead?'  
> ^C
```

```
$ echo -e "hello \nworld"
```

Linux Directory Structure

- ❖ Files on a Linux system are stored within a single directory structure, called a **virtual directory**.
- ❖ The virtual directory contains files from all the computer's storage devices and merges them into a **single directory structure**.
- ❖ This structure has a single base directory called the **root directory**, often simply called root.
- ❖ When you log into the Linux system, your process's current working directory is your **account's home directory**.
- ❖ A **current working directory** is the directory your process is currently using within the virtual directory structure.
- ❖ Think of the current working directory as the room you are currently in within your home.
- ❖ You can navigate through this virtual directory structure.

Linux Directory Tree



Navigating the Directory Structure

❖ **pwd** - print name of current/working directory

pwd [OPTION]...

✓ **-L, --logical**

- use PWD from environment, even if it contains symlinks

✓ **-P, --physical**

- avoid all symlinks

Navigating the Directory Structure

❖ **cd** - Change the shell working directory.

cd [OPTION]...

- ✓ **-L** force symbolic links to be followed
- ✓ **-P** use the physical directory structure without following symbolic links
- ✓ **-** return to your most recent working directory

Using Absolute and Relative File References

❖ Absolute References

❖ Relative References

✓ .

- refers to the current working directory.

✓ ..

- represent the directory above the current directory, which is the parent directory.

❖ Home Directory References

\$ cd

\$ cd ~

\$ cd \$HOME

Navigating the Directory Structure

```
$ pwd  
/home/anisa
```

```
$ cd /etc  
$ pwd  
/etc
```

```
$ cd cups  
$ pwd  
/etc/cups
```

```
$ cd ..  
$ pwd  
/etc
```

```
$ cd /var
```

```
$ pwd  
/var
```

```
$ cd -  
/etc
```

```
$ pwd  
/etc
```

```
$ cd  
$ pwd  
/home/anisa
```

Internal and External Commands

- ❖ Within a shell, some commands that you type at the command line are part of (internal to) the shell program.
 - ✓ These internal commands are sometimes called built-in commands.
- ❖ Other commands are external programs, because they are not part of the shell.

```
$ type echo
```

```
echo is a shell builtin
```

```
$ type pwd
```

```
pwd is a shell builtin
```

```
$ type uname
```

```
uname is /usr/bin/uname
```

Using Environment Variables

❖ **Environment variables** track specific system information, such as the name of the user logged into the shell, the default home directory for the user, the search path the shell uses to find executable programs, and so on.

```
$ set
```

```
$ env
```

```
$ printenv
```


Environment Variables

Name	Description
BASH_VERSION	Current Bash shell instance's version number
GROUPS	User account's group memberships
HISTSIZE	Maximum number of commands stored in history file
HOME	Current user's home directory name
HOSTNAME	Current system's host name
LANG	Locale category for the shell
PATH	Colon-separated list of directories to search for commands
PS1	Primary shell command-line interface prompt string
PS2	Secondary shell command-line interface prompt string
PWD	User account's current working directory
SHLVL	Current shell level
TZ	User's time zone, if different from system's time zone
UID	User account's user identification number
VISUAL	Default screen-based editor used by some shell commands

Locate Command

- ❖ When you enter a program name (command) at the shell prompt, the shell will search all the directories listed in the **PATH** environment variable for that program.
- ❖ If the shell cannot find the program, you will receive a **command not found error** message.

Locate Command

❖ **which** - locate a command

which [-a] filename ...

✓ **-a** print all matching pathnames of each argument

```
$ which Hello.sh
```

```
/usr/bin/which: no Hello.sh in  
(/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/Christine/.local/bin:/home/Christine/bin)
```

```
$ which echo
```

```
/usr/bin/echo
```

```
$ /usr/bin/echo Hello World
```

```
Hello World
```

Modify Environment Variables

❖ Using (=)

- ✓ not survive entering into a subshell

```
$ MYVAR=101
$ echo $SHLVL
1
$ echo $MYVAR
101
$ bash
$ echo $MYVAR
$
$ echo $SHLVL
2
$ exit
exit
```

❖ Using **export**

- ✓ survive entering into a subshell

```
$ export MYVAR=102
$ echo $MYVAR
102
$ bash
$ echo $SHLVL
2
$ echo $MYVAR
102
$ MYVAR="100"
$ export MYVAR
```

Unset Environment Variable

```
$ echo $EDITOR
```

```
$ export EDITOR=nano
```

```
$ echo $EDITOR
```

```
nano
```

```
$ unset EDITOR
```

```
$ echo $EDITOR
```

```
$
```


Getting Help

❖ **man** - an interface to the on-line reference manuals

man [OPTION] command

✓ **-k** , (Equivalent to **apropos command**)

- Search the short descriptions and manual page names for the keyword as regular expression.

✓ **-f** , (Equivalent to **what is command**)

- Lookup the manual pages referenced by keyword and print out the short descriptions of any found.

man Sections

❖ The collection of all man pages on a system is called the Linux Manual. The Linux manual is divided into sections, each of which covers a particular topic, and every man page is associated with exactly one of these sections.

❖ The sections are:

1. User Commands
2. System Calls
3. Library Calls
4. Device files (usually stored in /dev)
5. File Formats
6. Games
7. Miscellaneous (macro packages, conventions, and so on)
8. Administrative Commands
9. Kernel routines

See specific man page section:

```
$ man -S 5 passwd
```

```
$ man -s 5 passwd
```

```
$ man 5 passwd
```

History

- ❖ The shell keeps track of all the commands you have recently used and stores them in your login session's history list.
- ❖ **history** - List recent commands executed
 - ✓ -a
 - appends the current history list commands to the end of the history file.
 - ✓ -n
 - appends the history file commands from the current Bash shell session to the current history list.
 - ✓ -r
 - overwrites the current history list commands with the commands stored in the history file.
 - ✓ -W
 - write the current history to the history file

History

❖ \$ echo \$HISTFILE

- ✓ Viewing the history file path

❖ \$ history n

- ✓ lists the last N commands

❖ !!

- ✓ Re-execute your most recent command

❖ !n

- ✓ Re-execute command with number **n** in history list

Clean History

1. `history -c`

- ✓ clear your current history list

2. `history -w`

- ✓ wipe the history file

EDITING TEXT FILES

Manipulating text is performed on a regular basis when managing a Linux system.

Whether you need to modify a configuration file or create a shell script, being able to use an interactive text file editor at the command line is an important skill.



Looking at Text Editors

❖ emacs

❖ nano

❖ vim

Dealing with Default Editors

❖ You can change your account's standard editor via the **EDITOR** environment variables.

✓ `$ export EDITOR=nano`

vim versus vi

- ❖ The vi editor was a Unix text editor, and when it was rewritten as an open source tool, it was improved. Thus, vim stands for “vi improved.”
- ❖ Often you'll find the vi command will start the vim editor.

```
$ which vim
/usr/bin/vim
$ which vi
alias vi='vim'
/usr/bin/vim
```

On CentOS

```
type vi
vi is /usr/bin/vi
type vim
vim is hashed (/usr/bin/vim)
readlink -f /usr/bin/vi
/usr/bin/vim.basic ✓
/usr/bin/vim.tiny x
```

On Ubuntu



Understanding vim Modes

❖ The vim editor has three standard modes as follows:

✓ Command Mode

✓ Insert Mode

✓ Ex Mode

Command Mode

- ❖ This is the mode vim uses when you first enter the buffer area
- ❖ It is sometimes called normal mode.
- ❖ Here you enter keystrokes to enact commands.
- ❖ Command is the best mode to use for quickly moving around the buffer area.

Commonly used vim command mode - moving commands

- ❖ **h** Move cursor left one character.
- ❖ **l** Move cursor right one character.
- ❖ **j** Move cursor down one line (the next line in the text).
- ❖ **k** Move cursor up one line (the previous line in the text).
- ❖ **w** Move cursor forward one word to front of next word.
- ❖ **e** Move cursor to end of current word.
- ❖ **b** Move cursor backward one word.
- ❖ **^** Move cursor to beginning of line.
- ❖ **\$** Move cursor to end of line.
- ❖ **gg** Move cursor to the file's first line.
- ❖ **G** Move cursor to the file's last line.
- ❖ **nG** Move cursor to file line number n.
- ❖ **Ctrl+B** Scroll up almost one full screen.
- ❖ **Ctrl+F** Scroll down almost one full screen.
- ❖ **Ctrl+U** Scroll up half of a screen.
- ❖ **Ctrl+D** Scroll down half of a screen.
- ❖ **Ctrl+Y** Scroll up one line.
- ❖ **Ctrl+E** Scroll down one line

Commonly used vim command mode - editing commands

- ❖ **a** Insert text after cursor.
- ❖ **A** Insert text at end of text line.
- ❖ **dd** Delete current line.
- ❖ **dw** Delete current word.
- ❖ **i** Insert text before cursor.
- ❖ **I** Insert text before beginning of text line.
- ❖ **ZZ** Write buffer to file and quit editor.
- ❖ **o** Open a new text line below cursor, and move to insert mode.
- ❖ **O** Open a new text line above cursor, and move to insert mode.
- ❖ **p** Paste copied text after cursor.
- ❖ **P** Paste copied (yanked) text before cursor.
- ❖ **yw** Yank (copy) current word.
- ❖ **yy** Yank (copy) current line.
- ❖ **Alt+U** undo last modification

Commonly used vim command mode - editing commands

❖ COMMAND [NUMBER-OF-TIMES] ITEM

- ✓ **d-3-w** delete three words
- ✓ **d-3-d** delete three lines
- ✓ **d-G** delete to end of file
- ✓ **Y-3** copy (yank) the 3 lines from the cursor
- ✓ **Y-\$** copy (yank) the text from the cursor to the end of the text line

vim command mode - Searching

❖? Start a backward search

❖/ Start a forward search.

✓ The keystroke will display at the vim editor's bottom and allow you to type the text to find. If the first item

❖n Move to the next matching text pattern.

❖N Move to the previous matching text pattern

Insert Mode

- ❖ Insert mode is also called edit or entry mode.
- ❖ This is the mode where you can perform simple editing.
- ❖ There are not many commands or special mode keystrokes.
- ❖ You enter this mode from command mode by pressing the “i” key.
- ❖ At this point, the message `--Insert--` will display in the message area.
- ❖ You leave this mode by pressing the Esc key

Ex Mode

- ❖ This mode is sometimes also called colon commands because every command entered here is preceded with a colon (:).

Commonly used vim Ex mode commands

❖ **:! command**

- ✓ Execute shell command and display results, but don't quit editor.

❖ **:r! command**

- ✓ Execute shell command and include the results in editor buffer area.

❖ **:r file**

- ✓ Read file contents and include them in editor buffer area.

Saving changes in the vim text editor

- ❖ **:x** Write buffer to file and quit editor.
- ❖ **:wq** Write buffer to file and quit editor.
- ❖ **:wq!** Write buffer to file and quit editor (overrides protection).
- ❖ **:w** Write buffer to file and stay in editor.
- ❖ **:w!** Write buffer to file and stay in editor (overrides protection).
- ❖ **:q** Quit editor without writing buffer to file.
- ❖ **:q!** Quit editor without writing buffer to file (overrides protection).



PROCESSING TEXT USING FILTERS

At the Linux command line, you often need to view files or portions of them. In addition, you may need to employ tools that allow you to gather data chunks or file statistics for troubleshooting or analysis purposes. The utilities in this section can assist in all these activities.



File-Combining Commands - cat

❖ **cat** - concatenate files and print on the standard output

`cat [OPTION]... [FILE]...`

✓ `-A, --show-all`

equivalent to `-vET`

✓ `-E, --show-ends`

display \$ at end of each line

✓ `-n, --number`

number all output lines

✓ `-s, --squeeze-blank`

suppress repeated empty output lines

✓ `-T, --show-tabs`

display TAB characters as `^I`

✓ `-v, --show-nonprinting`

use `^` and `M-` notation, except for `LFD` and `TAB`

❖ There are interesting variants of the cat command - **bzcat**, **xzcat**, and **zcat**.

These utilities are used to display the contents of compressed files.

File-Combining Commands - cat

```
$ cat numbers.txt
```

```
42  
2A  
52  
0010 1010  
*
```

```
$ cat random.txt
```

```
42  
Flat Land  
Schrodinger's Cat  
0010 1010  
0000 0010
```

```
$ cat numbers.txt random.txt
```

```
42  
2A  
52  
0010 1010  
*  
42  
Flat Land  
Schrodinger's Cat  
0010 1010  
0000 0010
```

File-Combining Commands - paste

❖ **paste** - merge lines of files

```
paste [OPTION]... [FILE]...
```

```
$ cat numbers.txt
```

```
42
```

```
2A
```

```
52
```

```
0010 1010
```

```
*
```

```
$ cat random.txt
```

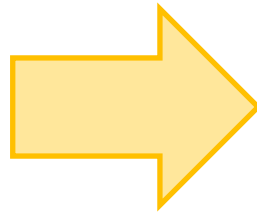
```
42
```

```
Flat Land
```

```
Schrodinger's Cat
```

```
0010 1010
```

```
0000 0010
```



```
$ paste random.txt numbers.txt
```

```
42 42
```

```
Flat Land 2A
```

```
Schrodinger's Cat 52
```

```
0010 1010 0010 1010
```

```
0000 0010 *
```

File-Transforming Commands - od

- ❖ Occasionally you may need to do a little detective work with files.
- ❖ These situations may include trying to review a graphics file or troubleshooting a text file that has been modified by a program.
- ❖ The **od** utility can help, because it allows you to display a file's contents in octal (base 8), hexadecimal (base 16), decimal (base 10), and ASCII.

File-Transforming Commands - od

❖ **od** - dump files in octal and other formats

od [OPTION]... [FILE]...

- ✓ The first column of the od command's output is an index number for each displayed line.
- ✓ **-c** select printable characters or backslash escapes
- ✓ **-b** select octal bytes

File-Combining Commands - od

```
$ cat fortytwo.txt
```

```
42  
fourty two  
quarante deux  
zweiundvierzig  
forti to
```

```
$ od fortytwo.txt
```

```
0000000 031064 063012 072557 072162 020171 073564 005157 072561  
0000020 071141 067141 062564 062040 072545 005170 073572 064545  
0000040 067165 073144 062551 075162 063551 063012 071157 064564  
0000060 072040 005157  
0000064
```

```
$ echo -e "Fanavaran \rAnisa" > anisa.txt
```

```
$ cat anisa.txt
```

```
Anisaaran
```

```
$ od -c anisa
```

```
0000000  f  a  n  a  v  a  r  a  n      \r  a  n  i  s  a  
0000020  \n  
0000021
```

Separating with split

❖ **split** - split a file into pieces

`split [OPTION]... [FILE [PREFIX]]`

✓ This utility allows you to divide a large file into smaller chunks, which is handy when you want to quickly create a smaller text file for testing purposes.

✓ `-l, --lines=NUMBER`

- put NUMBER lines/records per output file

Separating with split

```
$ cat fortytwo.txt
```

```
42
```

```
fourty two
```

```
quarante deux
```

```
zweiundvierzig
```

```
forti to
```

```
$ split -l 3 fortytwo.txt split42
```

```
$ ls split42*
```

```
split42aa split42ab
```

```
$ cat split42aa
```

```
42
```

```
fourty two
```

```
quarante deux
```

```
$ cat split42ab
```

```
zweiundvierzig
```

```
forti to
```

File-Formatting Commands - sort

❖ **sort** - sort lines of text files

sort [OPTION]... [FILE]...

✓ makes no changes to the original file; only the output is sorted.

✓ **-n, --numeric-sort**

- compare according to string numerical value

✓ **-r, --reverse**

- reverse the result of comparisons

✓ **-f, --ignore-case**

- fold lower case to upper case characters

✓ **-o file**

- Save on file

✓ **-u**

- (unique) removes duplicate lines in output

✓ **-t c**

- uses c as a field separator

✓ **-k X**

- sorts by c-delimited field X

File-Formatting Commands - nl

❖ **nl** - number lines of files

nl [OPTION]... [FILE]...

- ✓ This little command allows you to number lines in a text file in powerful ways.
- ✓ it will number only non-blank text lines
- ✓ **-ba** number all text file lines

File-Viewing Commands - more

❖ **more** - file perusal filter for crt viewing

`more [options] file...`

- ✓ **pressing the spacebar** move forward through a text file by one page down
- ✓ **pressing the Enter key** move forward one line down
- ✓ **press the q key** exit from the more pager
- ✓ **You cannot move backward through a file.**

File-Viewing Commands - less

❖ **less** - opposite of more

`less [options] file...`

- ✓ **? Or /** search through file
- ✓ **Esc+V** move back a page
- ✓ allows you to move backward



Looking at files with head

❖ **head** - output the first part of files

`head [options] file...`

- ✓ By default, the head command displays the first 10 lines of a text file.
- ✓ `-n, --lines=[-]NUM`
 - print the first NUM lines instead of the first 10;
 - with the leading '-', print all but the last NUM lines of each file

Viewing Files with tail

❖ **tail** - output the last part of files

tail [options] file...

✓ By default, the tail command displays the last 10 lines of a text file.

✓ **-f, --follow**

- output appended data as the file grows
- Watching new messages as they are added is very handy

✓ **-n, --lines=[+]NUM**

- output the last NUM lines, instead of the last 10;
- use -n +NUM to output starting with line NUM

File-Viewing Commands – head and tail

```
$ head /etc/passwd
```

```
$ head -n 2 /etc/passwd
```

```
$ head -2 /etc/passwd
```

```
$ head -n -2 /etc/passwd
```

```
$ tail /etc/passwd
```

```
$ tail -n 2 /etc/passwd
```

```
$ tail -2 /etc/passwd
```

```
$ tail -n +2 /etc/passwd
```

```
$ tail +2 /etc/passwd
```

Counting with wc

❖ **wc** - print newline, word, and byte counts for each file

wc [OPTION]... [FILE]...

✓ **-c, --bytes**

- Display the file's byte count.

✓ **-L, --max-line-length**

- Display the byte count of the file's longest line.

✓ **-l, --lines**

- Display the file's line count.

✓ **-m, --chars**

- Display the file's character count.

✓ **-w, --words**

- Display the file's word count.

```
$ wc random.txt
```

```
5 9 52 random.txt
```

Pulling Out Portions with cut

- ❖ To sift through the data in a large text file, it helps to quickly extract small data sections.
- ❖ The cut utility is a handy tool for doing this. It will allow you to view particular fields within a file's records.

Pulling Out Portions with cut

❖ Text File Records

- ✓ A text file record is a single-file line that ends in a newline linefeed, which is the ASCII character LF.
- ✓ You can see if your text file uses this end-of-line character via the `cat -E` command.
- ✓ It will display every newline linefeed as a \$.
- ✓ If your text file records end in the ASCII character NUL, you can also use `cut` on them, but you must use the `-z` option.

Pulling Out Portions with cut

❖ Text File Record Delimiter

- ✓ For some of the cut command options to be properly used, fields must exist within each text file record.
- ✓ These fields are not database-style fields but instead data that is separated by some delimiter.
- ✓ A delimiter is one or more characters that create a boundary between different data items within a record.
- ✓ A single space can be a delimiter.
- ✓ The password file, /etc/passwd, uses colons (:) to separate data items within a record.

Pulling Out Portions with cut

❖ Text File Changes

- ✓ Contrary to its name, the cut command does not change any data within the text file.
- ✓ It simply copies the data you wish to view and displays it to you. Rest assured that no modifications are made to the file.

Pulling Out Portions with cut

❖ **cut** - remove sections from each line of files

cut **OPTION...** [**FILE**]...

- ✓ **-c nlist, --characters nlist** Display only the record characters in the nlist (e.g., 1-5).
- ✓ **-b blist, --bytes blist** Display only the record bytes in the blist (e.g., 1-2).
- ✓ **-d d, --delimiter d** Designate the record's field delimiter as d. This overrides the Tab default delimiter. Put d within quotation marks to avoid unexpected results.
- ✓ **-f flist, --fields flist** Display only the record's fields denoted by flist (e.g., 1,3).
- ✓ **-s, --only-delimited** Display only records that contain the designated delimiter.
- ✓ **-z, --zero-terminated** Designate the record end-of-line character as the ASCII character NUL.

```
$ cut -d ":" -f 1,7 /etc/passwd
```

Discovering Repeated Lines with uniq

❖ **uniq** - report or omit repeated lines

```
uniq [OPTION]... [INPUT [OUTPUT]]
```

✓ **-c, --count**

- prefix lines by the number of occurrences

```
$ cat NonUniqueLines.txt
```

```
A
```

```
C
```

```
C
```

```
A
```

```
$ uniq NonUniqueLines.txt
```

```
A
```

```
C
```

```
A
```

Digesting an MD5 Algorithm

❖ **md5sum** - compute and check MD5 message digest

md5sum [OPTION]... [FILE]...

- ✓ The **md5sum** utility is based on the MD5 message-digest algorithm.
 - It was originally created to be used in cryptography.
 - It is no longer used in such capacities due to various known vulnerabilities.
 - However, it is still excellent for checking a file's integrity.
- ✓ The **md5sum** produces a 128-bit hash value.
 - If you copy the file to another system on your network, run the **md5sum** on the copied file.
 - If you find that the hash values of the original and copied file match, this indicates no file corruption occurred during its transfer.

Securing Hash Algorithms

- ❖ The Secure Hash Algorithms (SHA) is a family of various hash functions.
- ❖ Though typically used for cryptography purposes, they can also be used to verify a file's integrity after it is copied or moved to another location.

```
$ ls -l /usr/bin/sha???sum
```

```
/usr/bin/sha224sum
```

```
/usr/bin/sha256sum
```

```
/usr/bin/sha384sum
```

```
/usr/bin/sha512sum
```

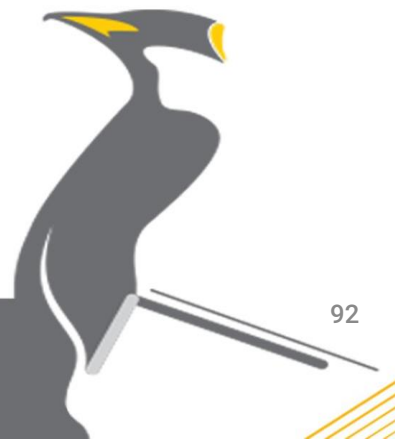
```
$ sha256sum fortytwo.txt
```

```
0b2b6e2d8eab41e73baf0961ec707ef98978bcd8c7
```

```
74ba8d32d3784aed4d286b fortytwo.txt
```

USING REGULAR EXPRESSIONS

Many commands use regular expressions. A regular expression is a pattern template you define for a utility such as `grep`, which then uses the pattern to filter text. Employing regular expressions along with text-filtering commands expands your mastery of the Linux command line.



Using grep

❖ **grep** - print lines that match patterns

grep [OPTION...] PATTERNS [FILE...]

- ✓ The grep command is powerful in its use of regular expressions, which will help with filtering text files.
- ✓ The grep command returns each file record (line) that contains an instance of the PATTERN
- ✓ **-c --count**
 - Display a count of text file records that contain a PATTERN match.
- ✓ **-d action --directories=action**
 - When a file is a directory, if action is set to read, read the directory as if it were a regular text file;
 - If action is set to skip, ignore the directory;
 - If action is set to recurse, act as if the -R, -r, or --recursive option was used.
- ✓ **-i --ignore-case**
 - Ignore the case in the PATTERN as well as in any text file records.
- ✓ **-v --invert-match**
 - Display only text files records that do not contain a PATTERN match.
- ✓ **-f FILE, --file=FILE**
 - Obtain patterns from FILE, one per line.
 - The empty file contains zero patterns, and therefore matches nothing.

Using grep

✓ `grep -F, --fixed-strings, fgrep`

- Interpret PATTERNS as fixed strings, not regular expressions.

✓ `grep -E, --extended-regexp, egrep`

- Interpret PATTERNS as extended regular expressions

✓ `grep -r, --recursive, rgrep, grep -d recurse`

- Read all files under each directory, recursively, following symbolic links only if they are on the command line.
- Note that if no file operand is given, grep searches the working directory.

Using grep

```
$ grep root /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
```

```
operator:x:11:0:operator:/root:/sbin/nologin
```

```
$ cat accounts.txt
```

```
sshd
```

```
Christine
```

```
nfsnobody
```

```
$ fgrep -f accounts.txt /etc/passwd
```

```
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
```

```
Christine:x:1001:1001:~/home/Christine:/bin/bash
```

```
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
```

```
$ grep -F -f accounts.txt /etc/passwd
```

Basic Regular Expressions (BRE)

The most powerful basic regular expression features include the following:

- ❖ **Bracket Expressions:** `b[aeiou]g` matches the words bag, beg, big, bog, and bug. **(`b[^aeiou]g`)**
- ❖ **Range Expressions:** `a[2-4]z`, `a[a-z]z`
- ❖ **Any Single Character:** `(.)` represents any single character except a newline => `a.z`
- ❖ **Start and End of Line:** The caret (^) represents the start of a line, and the dollar sign (\$) denotes the end of a line
- ❖ **Repetition:** an asterisk (*) denotes zero or more matches => `A.*Lincoln`
 - ✓ `.*` = Represent multiple characters
- ❖ **Escaping:** precede it with a backslash (\)

```
$ grep daemon.*nologin /etc/passwd
daemon:x:2:2:daemon:/sbin:/sbin/nologin
[...]
daemon:/dev/null:/sbin/nologin
[...]

$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin

$ grep ^root /etc/passwd
root:x:0:0:root:/root:/bin/bash

$ grep -v nologin$ /etc/passwd
root:x:0:0:root:/root:/bin/bash
sync:x:5:0:sync:/sbin:/bin/sync
[...]
Christine:x:1001:1001::/home/Christine:/bin/bash
```

```
$ grep -v ^$ filename
```

filter out all the blank lines in a file (display only lines with text)



Character Classes

- ❖ **[[:alnum:]]** Matches any alphanumeric characters (any case), and is equal to using the `[0-9A-Za-z]` bracket expression
- ❖ **[[:alpha:]]** Matches any alphabetic characters (any case), and is equal to using the `[A-Za-z]` bracket expression
- ❖ **[[:blank:]]** Matches any blank characters, such as tab and space
- ❖ **[[:digit:]]** Matches any numeric characters, and is equal to using the `[0-9]` bracket expression
- ❖ **[[:lower:]]** Matches any lowercase alphabetic characters, and is equal to using the `[a-z]` bracket expression
- ❖ **[[:punct:]]** Matches punctuation characters, such as `!`, `#`, `$`, and `@`
- ❖ **[[:space:]]** Matches space characters, such as tab, form feed, and space
- ❖ **[[:upper:]]** Matches any uppercase alphabetic characters, and is equal to using the `[A-Z]` bracket expression

```
$ cat random.txt
```

```
42
```

```
Flat Land
```

```
Schrodinger's Cat
```

```
0010 1010
```

```
0000 0010
```

```
$ grep [[:digit:]] random.txt
```

```
42
```

```
0010 1010
```

```
0000 0010
```


Extended Regular Expressions (ERE)

Extended regular expressions add more features that you can use to match in additional ways:

- ❖ **Additional Repetition Operators:** a plus sign (+) matches one or more occurrences, and a question mark (?) specifies zero or one match
- ❖ **Multiple Possible Strings:** The vertical bar (|) separates two possible matches. => **car|truck** matches either car or truck
- ❖ **Parentheses:** Ordinary parentheses (()) surround subexpressions. Parentheses are often used to specify how operators are to be applied

```
$ grep -E "^root|^dbus" /etc/passwd
```

```
root:x:0:0:root:/root:/bin/bash
```

```
dbus:x:81:81:System message bus:/:/sbin/nologin
```

```
$ egrep "(daemon|s).*nologin" /etc/passwd
```

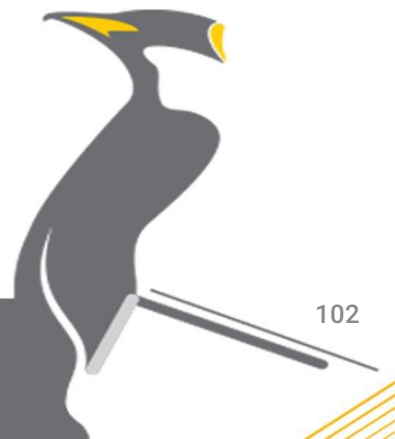
```
bin:x:1:1:bin:/bin:/sbin/nologin
```

```
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

```
[...]
```

USING STREAMS, REDIRECTION, AND PIPES

One of the neat things about commands at the command line is that you can employ complex frameworks. These structures allow you to build commands from other commands, use a program's output as input to another program, put together utilities to perform custom operations, and so on.



Standard Input and Output

❖ Linux provides three I/O channels to Programs

- ✓ Standard input (STDIN) - keyboard by default, **file descriptor = 0**
- ✓ Standard output (STDOUT) - terminal window by default, **file descriptor = 1**
- ✓ Standard error (STDERR) - terminal window by default, **file descriptor = 2**

❖ STDOUT and STDERR can be redirected to files:

- ✓ **command operator filename**
- ✓ Supported operators include:
 - **1>, >** Redirect STDOUT to file
 - **2>** Redirect STDERR to file
 - **&>** Redirect all output to file (STDOUT and STDERR)
 - **>>** appends (>>, 2>>, &>>)

Redirecting Output to a File

❖ Examples:

- ❖ This command generates output and errors when run as non-root:

```
$ find /etc -name passwd
```

- ❖ Operators can be used to store output and errors:

```
$ find /etc -name passwd > find.out
```

```
$ find /etc -name passwd 2> /dev/null
```

```
$ find /etc -name passwd > find.out 2> find.err
```

Combining Output and Errors

Some operators affect both STDOUT and STDERR

❖ **&>**: Redirects all output:

```
$ find /etc -name passwd &> find.all
```

❖ **2>&1**: Redirects STDERR to STDOUT

✓ Useful for sending all output through a pipe

```
$ find /etc -name passwd 2>&1 | less
```

❖ **()**: Combines STDOUTs of multiple programs

```
$ ( cal 2017 ; cal 2018 ) | less
```

Redirecting STDIN from a File

- ❖ Redirect standard input with <

- ❖ Some commands can accept data redirected to STDIN from a file:

```
$ tr 'A-Z' 'a-z' < .bash_profile
```

- ✓ This command will translate the uppercase characters in .bash_profile to lowercase

```
$ echo "Fanavaran Anisa" | tr [:space:] '\t'
```

- ❖ Equivalent to:

```
$ cat .bash_profile | tr 'A-Z' 'a-z'
```


Sending Multiple Lines to STDIN

❖ Redirect multiple lines from keyboard to STDIN with <<WORD

- ✓ All text until WORD is sent to STDIN
- ✓ Sometimes called a **heretext**

```
$ mail -s "Please Call" security@anisa.co.ir <<ANISA
> To whom it may concern,
> Please give me a call when you get in. We may need
> to do some maintenance on the server.
> Details when you're on-site,
> Amini
> ANISA
```

Sending Multiple Lines to STDIN

- ❖ `<>` Causes the specified file to be used for both standard input and standard output.

```
$ tr 'A-Z' 'a-z' <> .bash_profile
```

Piping Data between Programs

- ❖ Pipes (the | character) can connect commands:

`command1 | command2`

- ✓ Sends STDOUT of command1 to STDIN of command2
- ✓ STDERR is not forwarded across pipes

- ❖ Used to combine the functionality of multiple tools

✓ `command1 | command2 | command3... etc.`

Useful Pipe Targets

```
$ grep /bin/bash$ /etc/passwd | wc -l
```

```
$ grep /sbin/nologin$ /etc/passwd | cut -d ":" -f 1 | \
sort | less
```

```
$ sort filename | uniq -c
```

Redirecting to Multiple Targets (tee)

tee command reads the standard input and writes it to both the standard output and one or more files

```
$ command1 | tee filename | command2
```

- ✓ Stores STDOUT of **command1** in filename, then pipes to **command2**

❖ Uses:

- ✓ Troubleshooting complex pipelines
- ✓ Simultaneous viewing and logging of output

Using sed

❖ **sed** - stream editor for filtering and transforming text

`sed [OPTIONS] [SCRIPT]... [FILENAME]`

- ✓ A stream editor modifies text that is passed to it via a file or output from a pipeline.
- ✓ `-e script, --expression=script`
 - Add commands in script to text processing. The script is written as part of the sed command.
- ✓ `-f script --file=script`
 - Add commands in script to text processing. The script is a file.
- ✓ `-r --regexp-extended`
 - Use extended regular expressions in script.

Using sed

❖ Using sed to modify STDIN text

```
$ echo "I like cake." | sed 's/cake/donuts/'  
I like donuts.
```

❖ Using sed to globally modify STDIN text

```
$ echo "I love cake and more cake." | sed 's/cake/donuts/'  
I love donuts and more cake.  
$ echo "I love cake and more cake." | sed 's/cake/donuts/g'  
I love donuts and more donuts.
```

❖ Using sed -e to use multiple scripts

```
$ sed -e 's/cake/donuts/ ; s/like/love/' cake.txt  
Christine loves chocolate donuts.  
Rich loves lemon donuts.  
Tim only loves yellow donuts.  
Samantha does not love donuts.
```


Using sed

❖ Using sed to modify file text

```
$ cat cake.txt
```

```
Christine likes chocolate cake.
```

```
Rich likes lemon cake.
```

```
Tim only likes yellow cake.
```

```
Samantha does not like cake.
```

```
$ sed 's/cake/donuts/' cake.txt
```

```
Christine likes chocolate donuts.
```

```
Rich likes lemon donuts.
```

```
Tim only likes yellow donuts.
```

```
Samantha does not like donuts.
```

Using sed

❖ Using sed to delete file text

```
$ sed '/Christine/d' cake.txt
```

```
Rich likes lemon cake.
```

```
Tim only likes yellow cake.
```

```
Samantha does not like cake.
```

❖ Use the syntax of '**ADDRESScNEWTEXT**' for the sed command's SCRIPT. The ADDRESS refers to the file's line

```
$ sed '4cI am a new line' cake.txt
```

```
Christine likes chocolate cake.
```

```
Rich likes lemon cake.
```

```
Tim only likes yellow cake.
```

```
I am a new line
```

Generating Command Lines

❖ **xargs** - build and execute command lines from standard input

`xargs [options] [command [initial-arguments]]`

✓ `-p, --interactive`

- Prompt the user about whether to run each command line and read a line from the terminal.

```
$ touch EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt
```

```
$ ls EmptyFile?.txt
```

```
EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt
```

```
$ ls -1 EmptyFile?.txt | xargs -p /usr/bin/rm
```

```
/usr/bin/rm EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt ?...n
```

Generating Command Lines

❖ Using the `$()` method to create commands

```
$ rm -i $(ls EmptyFile?.txt)
```

```
rm: remove regular empty file 'EmptyFile1.txt'? y
```

```
rm: remove regular empty file 'EmptyFile2.txt'? y
```

```
rm: remove regular empty file 'EmptyFile3.txt'? Y
```

❖ Using the ``` method to create commands

```
$ rm -i `ls EmptyFile?.txt`
```

```
rm: remove regular empty file 'EmptyFile1.txt'? y
```

```
rm: remove regular empty file 'EmptyFile2.txt'? y
```

```
rm: remove regular empty file 'EmptyFile3.txt'? y
```