Linux Systems

Getting started with setting up

Overview of Linux Systems

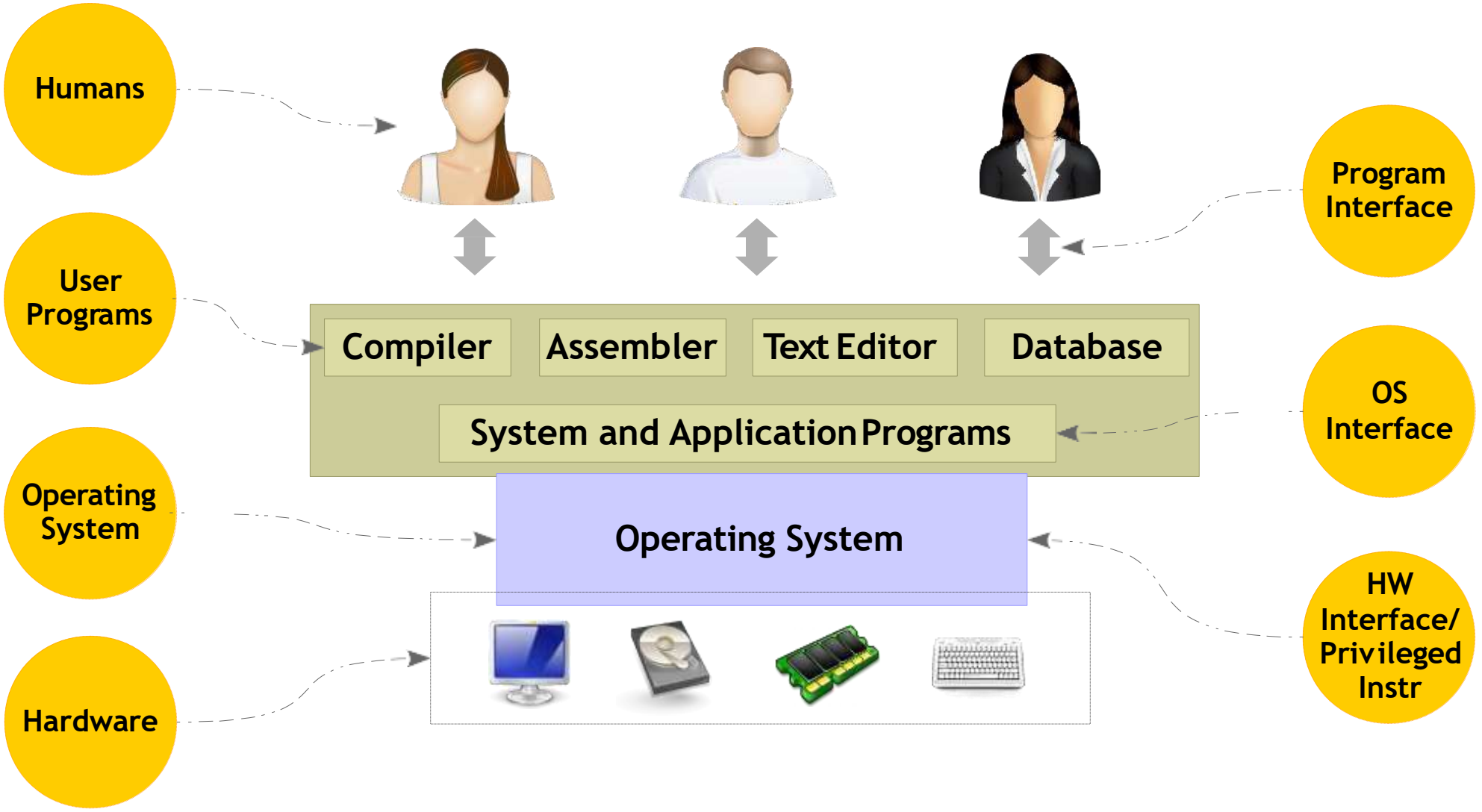
Linux Systems

Introduction - Let us ponder ...

- What exactly is an Operating System (OS)?
- Why do we need OS?
- How would the OS would look like?
- Is it possible for a team of us (in the room) to create an OS of our own?
- Is it necessary to have an OS running in a Embedded System?
- Will the OS ever stop at all?

Linux Systems

Introduction - Operating System



Linux Systems

Introduction - What is Linux?

- Linux is a free and open source operating system tha causing a revolution in the computer world
- Originally created by Linus Torvalds with the assistance of developers called community
- This operating system in only a few short years is beginning to dominate markets worldwide
- Today right from hand-held devices (ex: Android) to high end systems (ex: Stock exchange servers) use Linux

Linux Systems

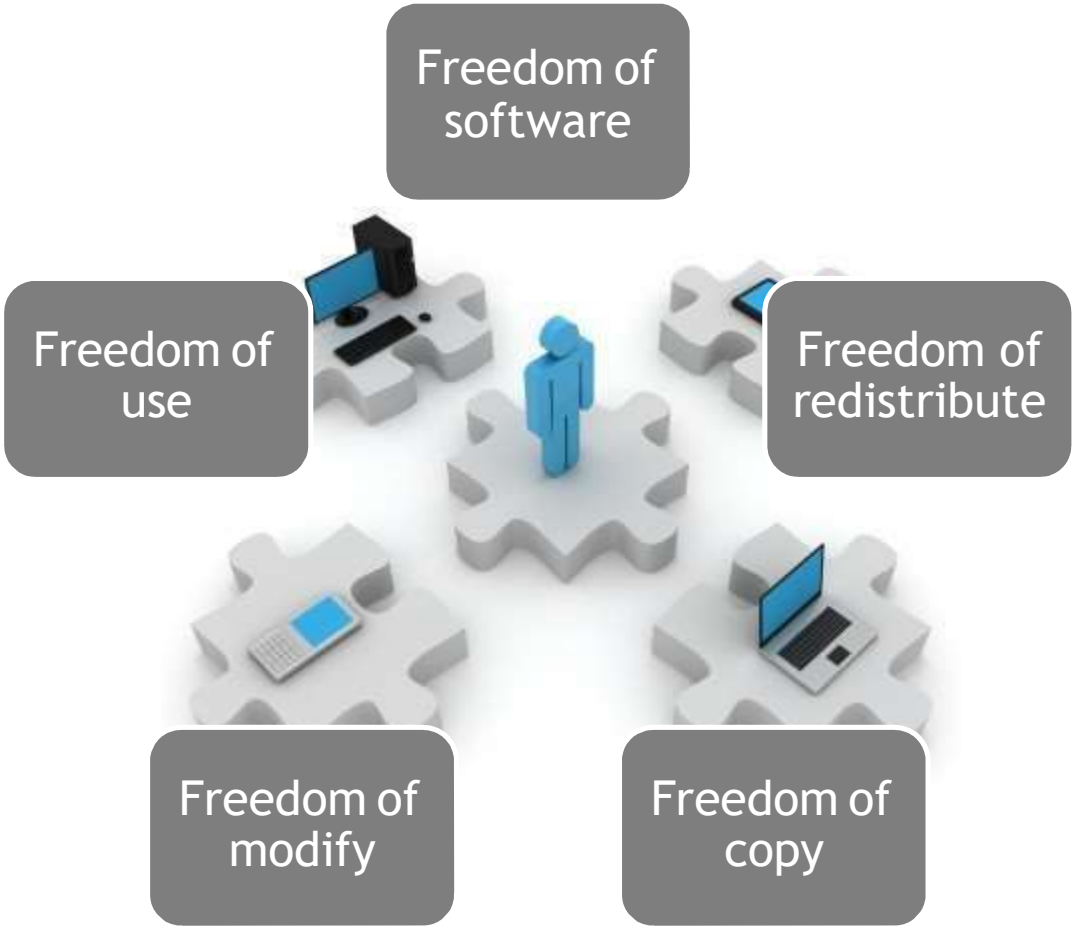
Introduction - Why use Linux?

- Free & Open Source –GPL license, no cost
- Reliability –Build systems with 99.999% upstream
- Secure –Monolithic kernel offering high security
- Scalability –From mobile phone to stock market servers

The word 'Free' in Open Source should be interpreted as in 'Freedom' not as 'Free Beer'. This also explains the spirit of creating Open Source software.

Linux Systems

Introduction - What is Open Source?



Linux Systems

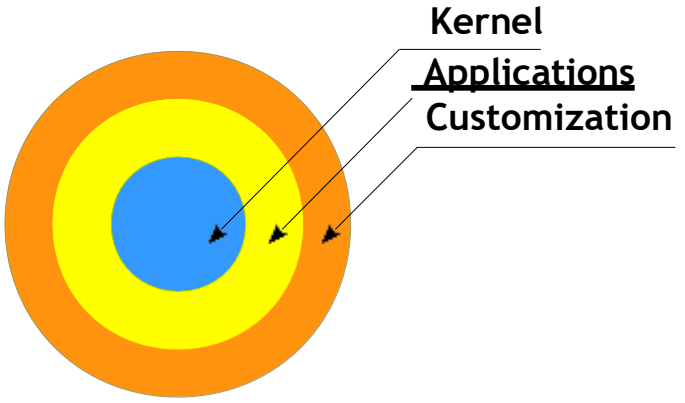
Introduction - Open Source - How it all started?

- With GNU (GNU is not UNIX)
- Richard Stallman made the initial announcement in 1983, Free Software Foundation (FSF) got formed during 1984
- Volunteer driven GNU started developing multiple projects, but making it as an operating system was always a challenge
- During 1991 a Finnish Engineer Linus Torvalds developed core OS functionality, called it as “Linux Kernel”
- Linux Kernel got licensed under GPL, which laid strong platform for the success of Open Source
- Rest is history!

Linux Systems

Introduction - Open Source - How it evolved?

- Multiple Linux distributions started emerging around the Kerne
- Some applications became platform independent
- Community driven software development started picking up
- Initially seen as a “geek-phenomenon”, eventually turned out to be an engineering marvel
- Centered around Internet
- Building a business around open source started becoming viable
- Redhat set the initial trend in the OS business



Linux Systems

Introduction - Open Source - Where it stands now?

OS

Databases

Server/Cloud

Enterprise

Consumer

Education

CMS

eCommerce

Linux Systems

Introduction - Open Source vs Freeware

OSS	Freeware
Users have the right to access & modify the source codes In case original programmer disappeared, users & developer group of the S/W usually keep its support to the S/W. OSS usually has the strong users & developers group that manage and maintain the project	Freeware is usually distributed in a form of binary at 'Free of Charge', but does not open source codes itself. Developer of freeware could abandon development at any time and then final version will be the last version of the freeware. No enhancements will be made by others. Possibility of changing its licensing policy

Linux Systems

Introduction - GPL

- Basic rights under the GPL – access to source code, r to make derivative works
- Reciprocity/Copy-left
- Purpose is to increase amount of publicly available software and ensure compatibility
- Licensees have right to modify use or distribute software, and to access the source code

Linux Systems

Introduction - GPL - Issues

- Linking to GPL programs
- No explicit patent grant
- Does no discuss trademark rights
- Does not discuss duration
- Silent on sub-licensing
- Relies exclusively on license law, not contract

Linux Systems

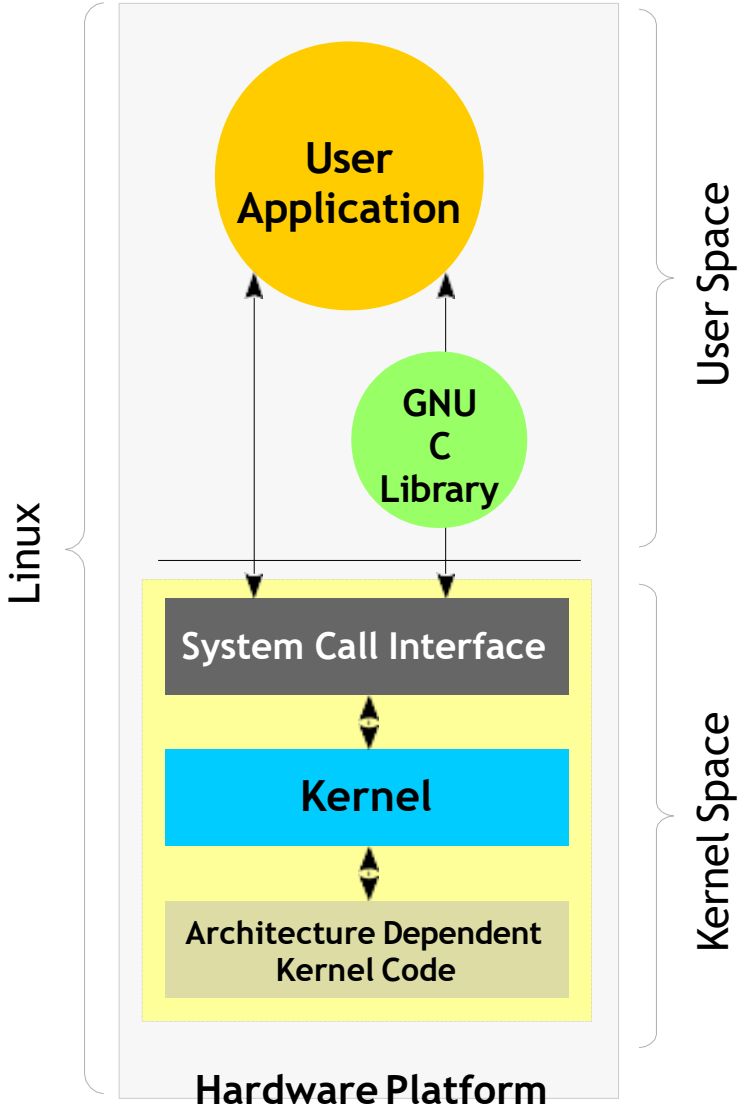
Introduction - Linux Properties

What has made Linux so popular to scale from mobile devices powering 90% of world's super computer? Here are the key properties of Linux

- Multitasking
  - Ability to handle multiple tasks across single / multiple processors
- Multi-user
  - Have got users with different level of privileges for secured access
- Protected Memory
  - Clear distinction called 'user-space' and 'kernel' space thereby having protected memory access. This makes Linux Super secure comparing with other operating systems
- Hierarchical File System
  - Well organized file system that handles various types of files. This also makes handling various inputs very simple

Linux Systems

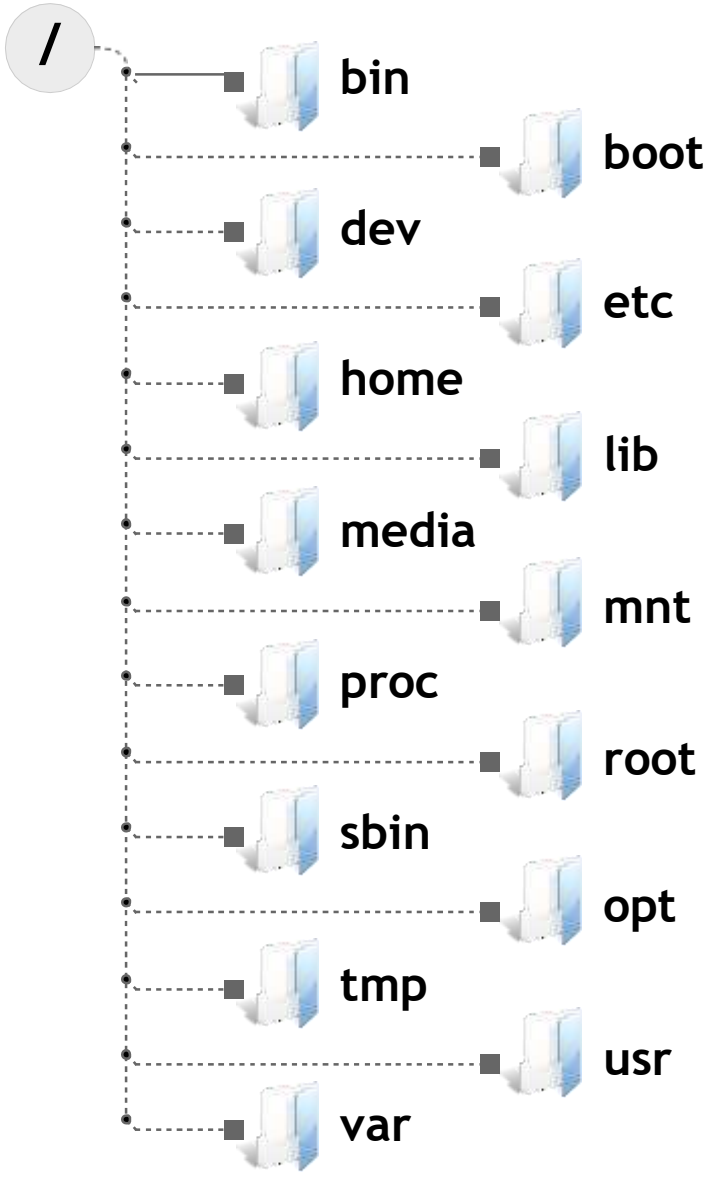
Introduction - Linux Components



- **Hardware Controllers:** This subsy comprised of all the possible physical devices in a Linux installation - CPU, memory hardware, hard disks
- **Linux Kernel:** The kernel abstracts and mediates access to the hardware resources, including the CPU. A kernel is the core of the operating system
- **O/S Services:** These are services that are typically considered part of the operating system (e.g. windowing system, command shell)
- **User Applications:** The set of applications in use on a particular Linux system (e.g. web browser)

Linux Systems

Introduction - Linux Directory Structure



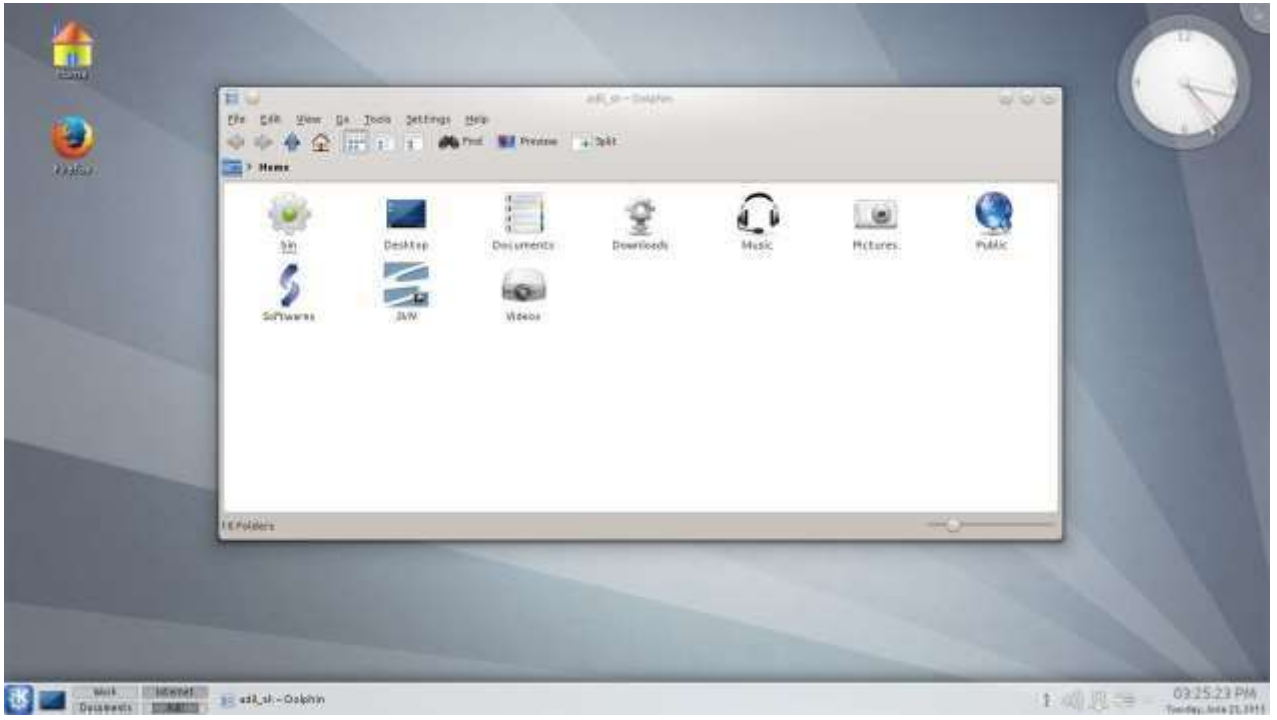
bin	Essential user command binaries
boot	Static boot-able images
dev	Device files
etc	Host specific configuration
home	User home directories
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for temporarily mounted file systems
proc	Virtual FS documenting kernel and process status
root	Root user's home directory
sbin	Essential super user command binaries
opt	Add-on application software packages
tmp	Temporary files
usr	Multi user utilities and application
var	Variable files (Logs)

User Interfaces

Linux Systems

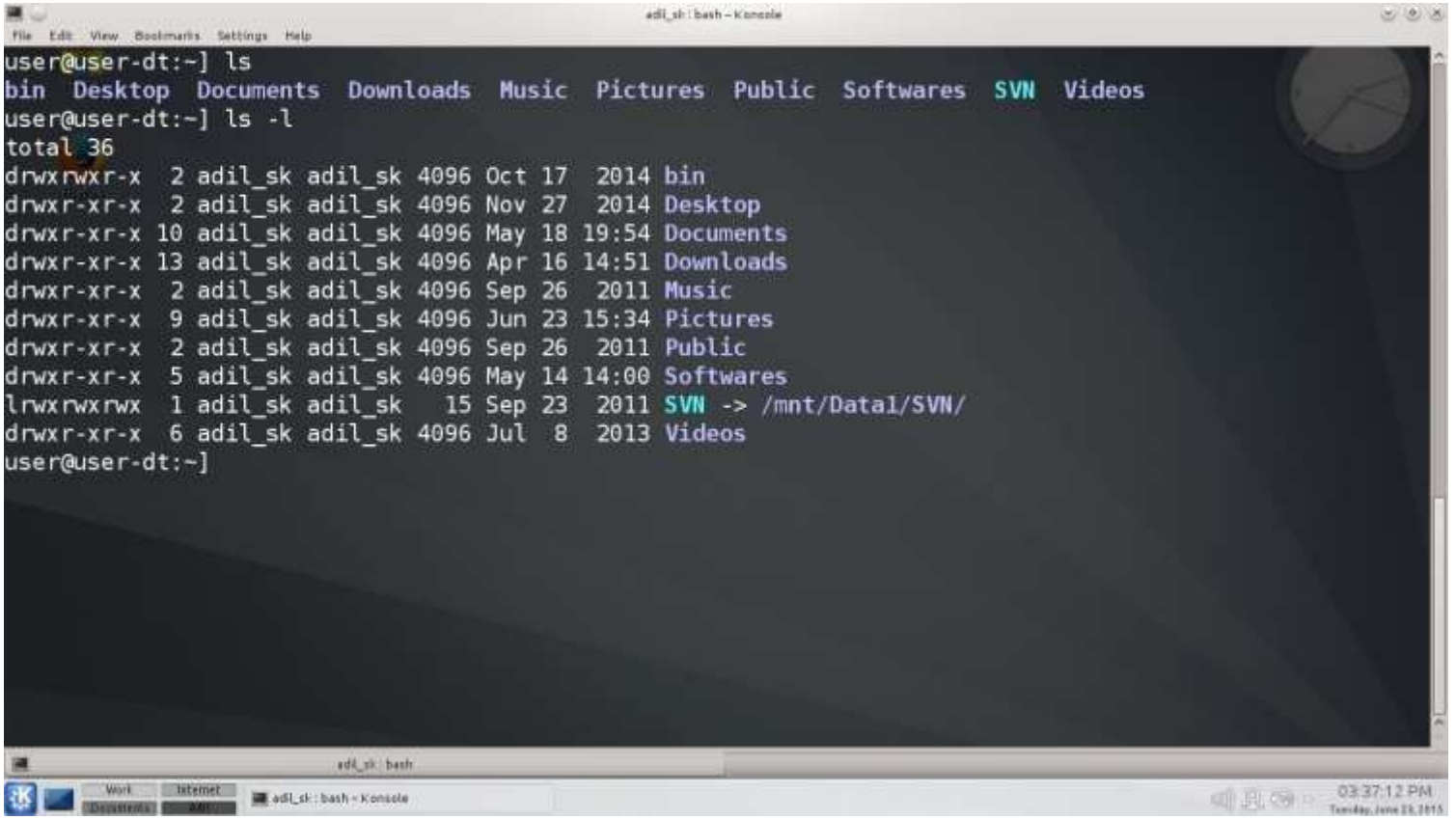
User Interface - GUI

- In graphical mode the user will be given a GUI using which he / she will be able to use the system using mouse
- Similar to windows based system that exist in other operating systems like MS Windows & Apple MAC OS



Linux Systems
User Interface - CLI

- Textual mode used to execute requested commands



Our focus is to be in the CLI mode by executing various commands by invoking shells. We will also create programs using this environment called 'Shell scripts'

Linux Systems

User Interface - The Shell - Introduction

- Shell is an application, works as a command interpreter
- Gets a command from user, gets it executed from OS
- Gives a programming environment to write scripts using interpreted language
- It has been inherited from UNIX operating system, which was predecessor to Linux

Linux Systems

User Interface - The Shell - Types

- Login
  - Starts after a successful login
  - It is executed under user ID during login process
  - It picks up user specific configuration and loads them

Linux Systems

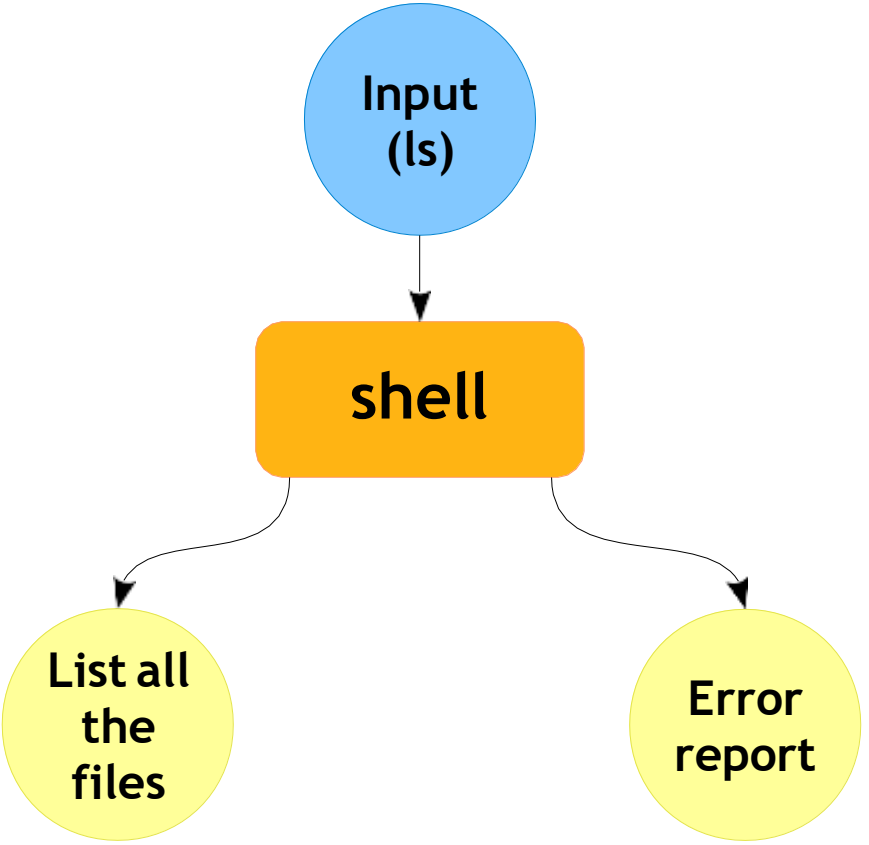
User Interface - The Shell - Types

- Non Login
  - A Non login shell is started by a program without a login
  - In this case, the program just passes the name of the shell executable
  - For example, for a Bash shell it will be simply bash
  - Following are examples of Non-login shells:
    - sh
    - bash
    - ksh
    - csh

Linux Systems

User Interface - The Shell - Invocation

- The main task of a shell is providing a user environment



Linux Systems

User Interface - The Shell - Bash

- Bash – The command interpreter
- GNU Project's shell
- Bash is the Bourne Again Shell
- Some features of bash are
  - Command line editing
  - Unlimited size command history
  - Job Control
  - Shell Functions and Aliases
  - Indexed arrays of unlimited size
  - Integer arithmetic in any base from two to sixty-four

Shell Usage and Basic Commands

Linux Systems

Basic Shell Commands – Points to be Noted

- It is assumed you follow the slides sequence, since s
  of the commands explained assumes, you have executed
  them before trying the next one
- After typing the command its expected you press an
  enter key

Linux Systems

Basic Shell Commands - pwd

- As soon as we open an terminal we are taken to the home directory
- This can be known with the **pwd** command

```
user@user:~] pwd
/home/user
user@user:~]
```

- Basically, the **pwd** command tells you about current working directory

Linux Systems

Basic Shell Commands - ls

- Well, we come to know where we are, how do we kr what do we have there?

```
user@user:~] ls
Desktop                 Pictures   Templates  Videos   hello.c
          Downloa       Public     Test1      bin      test2
ds  Documents
          Music
```

- The list command **ls** helps us here. The ls display the contents of the current directory.
- Well one question should pop up about the category of these files?
- From the above image, can tell which is a directory, file etc..?

Linux Systems

Basic Shell Commands - man

- So how do we come to know about different listing options?

```
user@user:~] man ls
```

- You may use **man** command, which stands for manual
- The most useful command which acts as reference manual if you work in Linux system
- Almost all the installed applications, libraries and all would have its own manual entry
- Even man has its own man page!

```
user@user:~] man man
```

- This helps us to know how read man pages and different sections in it

Linux Systems

Basic Shell Commands - man

- Hey, what? It looks too complex!! 
- Well, yes as starter most of the things looks complex, that too a technical documentation 
- Habit of reading it would certainly make it easy.

```
MAN(1) ⇐ This is a man page section, There is chance of a command
                                                                    MAN(1
                ) and libray function have a same name, so section identifies it

NAME
        The name of command / function is described here

SYNOPSIS
        How to use the command gets described here

DESCRIPTION
        Description of the command and function comes here

OTHER SUBSECTION ⇐ Based on the man page different sub section come
                   below here
```

Linux Systems

Basic Shell Commands - man

- So from **man** page of **ls** we get the following option

```
user@user:~] ls -F
Desktop/     Downloads/   Pictures/    Templates/   Videos/    hello.c
Documents/   Music/       Public/      Test1/       bin/       test2@
user@user:~]
```

- Where

```
/ →Directory Directory @
→Directory Symbolic Link
*      →Directory Executable
|      →Directory
Pipe  and
more
```

Linux Systems

Basic Shell Commands - Anatomy of a Command

```
user@user:~] command_name [arguments]
```

[*arguments*]

- A command may have multiple arguments
- Arguments could be ***options*** to the command, ***file paths*** or ***arguments*** itself
  options starts with **-** which is called as *short options* which has single letter or -- called as ***long options*** with a word
- Some of the arguments are **optional** which is mentioned within **[ ]**
- The below example shows the contents of Documents directory with color option enabled

```
user@user:~] ls -l --color Documents/
```

command_name

- A command which gets interpreted by shell
- Could be a **super user** command which gets executed with **sudo**
- The below is an example shows how to install a package in Ubuntu

```
user@user:~] sudo apt install vim
```

user@user:~]

- Command prompt, which could be customized
- **Default prompt** after installation would look like as shown below

```
user@user:~$
```

Linux Systems

Basic Shell Commands – Types of commands

- An executable program like all those files can have in /usr/bin.
- A command built into the shell itself. bash provides a number of commands internally called shell built-ins The cd command, for example, is a shell built-in
- A shell function. These are miniature shell scripts incorporated into the environment.
- An alias. Commands that you can define yourselves, built from other commands.

Linux Systems

Basic Shell Commands - Types of commands

- To know the type of a command, you may try the following

```
user@user:~] type <command_name>
```

- Few examples

```
user@user:~] type ls
ls is aliased to `ls -v --color=auto'
user@user:~] type pwd
pwd is a shell builtin
user@user:~] type clear
clear is /usr/bin/clear
user@user:~]
```

Linux Systems

Basic Shell Commands - cd

- **cd** to change directory. A shell built-in command

```
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures   Public   Templates   Videos
user@user:~] cd Documents/
user@user:Documents]
```

. The above example changes the directory to Documents

. Now how to go back?!

- We need to understand the concept of the path, which is explained in the next slide

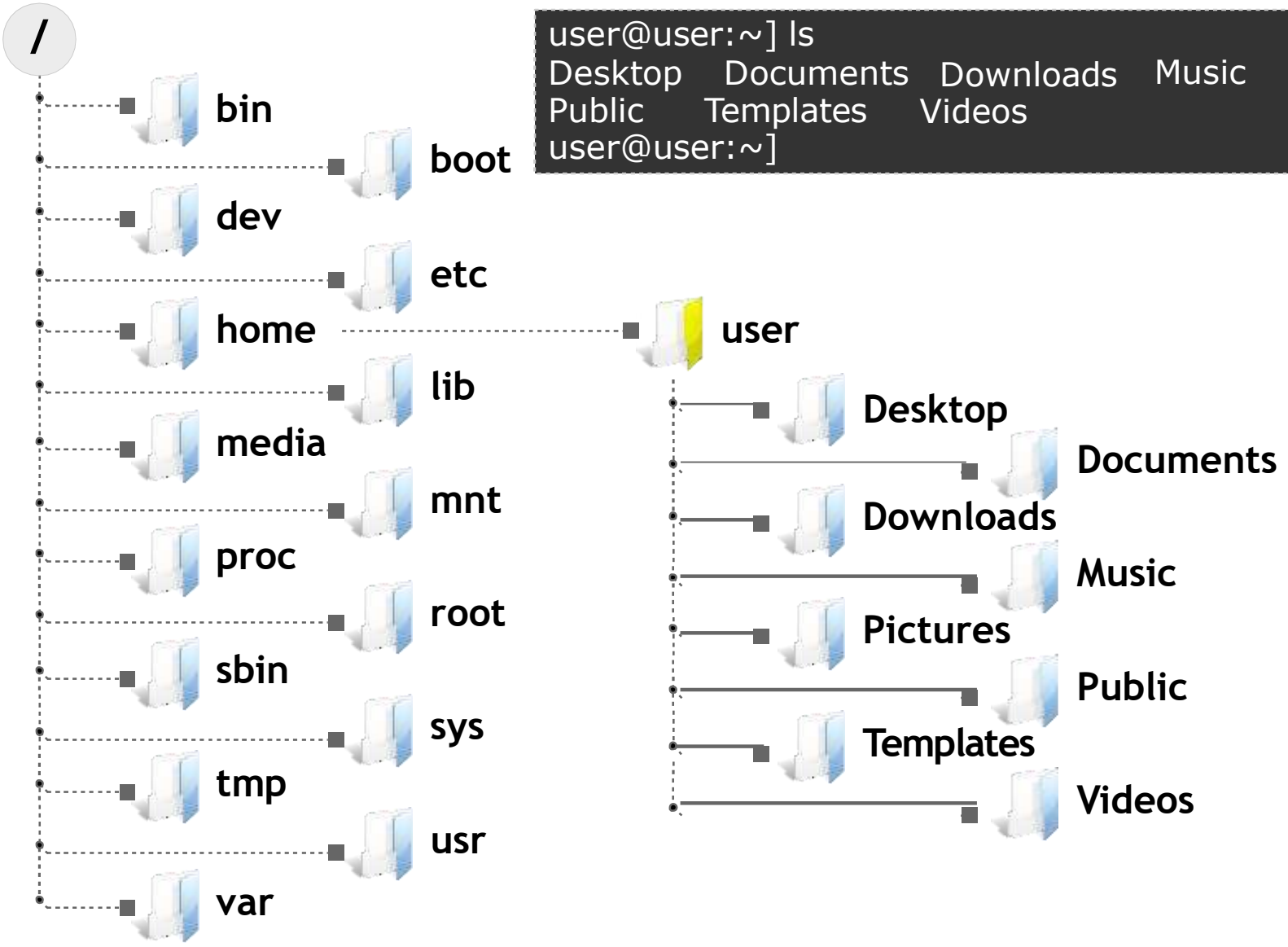
Linux Systems

Basic Shell Commands - Path

- Path is the location where a particular file is locate the directory (tree) structure
- It starts with the root (**'/'**) directory and goes into appropriate directory
- The path depends on the reference point from where you take it up:
  - **Absolute Path:** Specifies the location with reference from root directory
  - **Relative Path:** Specifies the location with reference to present working directory (**pwd**)
    - As the name says relative path will vary depending on your present working directory

Linux Systems

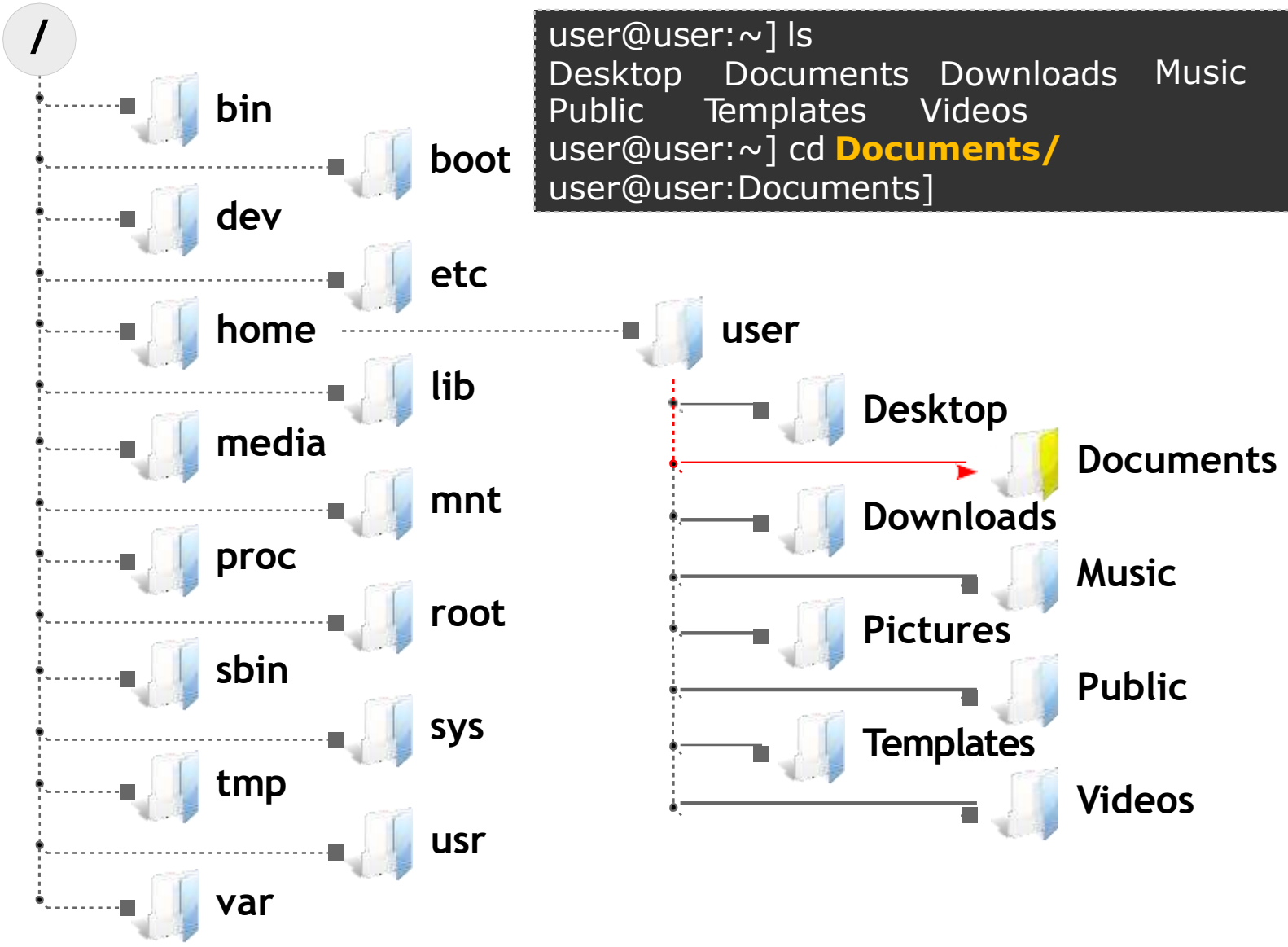
Basic Shell Commands - Path




```
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures
Public    Templates   Videos
user@user:~]
```

Linux Systems

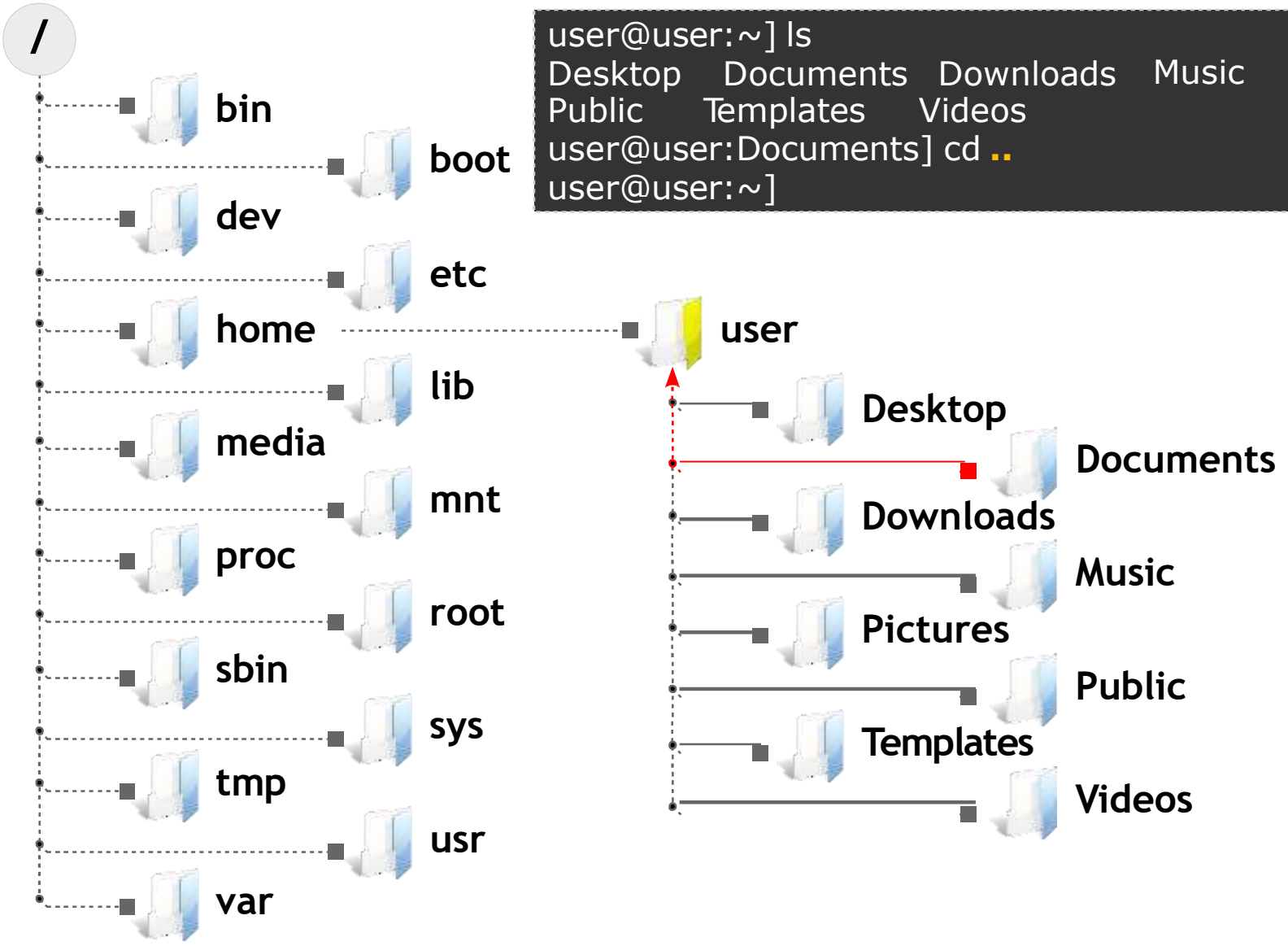
Basic Shell Commands - Path - Relative




```
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures
Public    Templates   Videos
user@user:~] cd Documents/
user@user:Documents]
```

Linux Systems

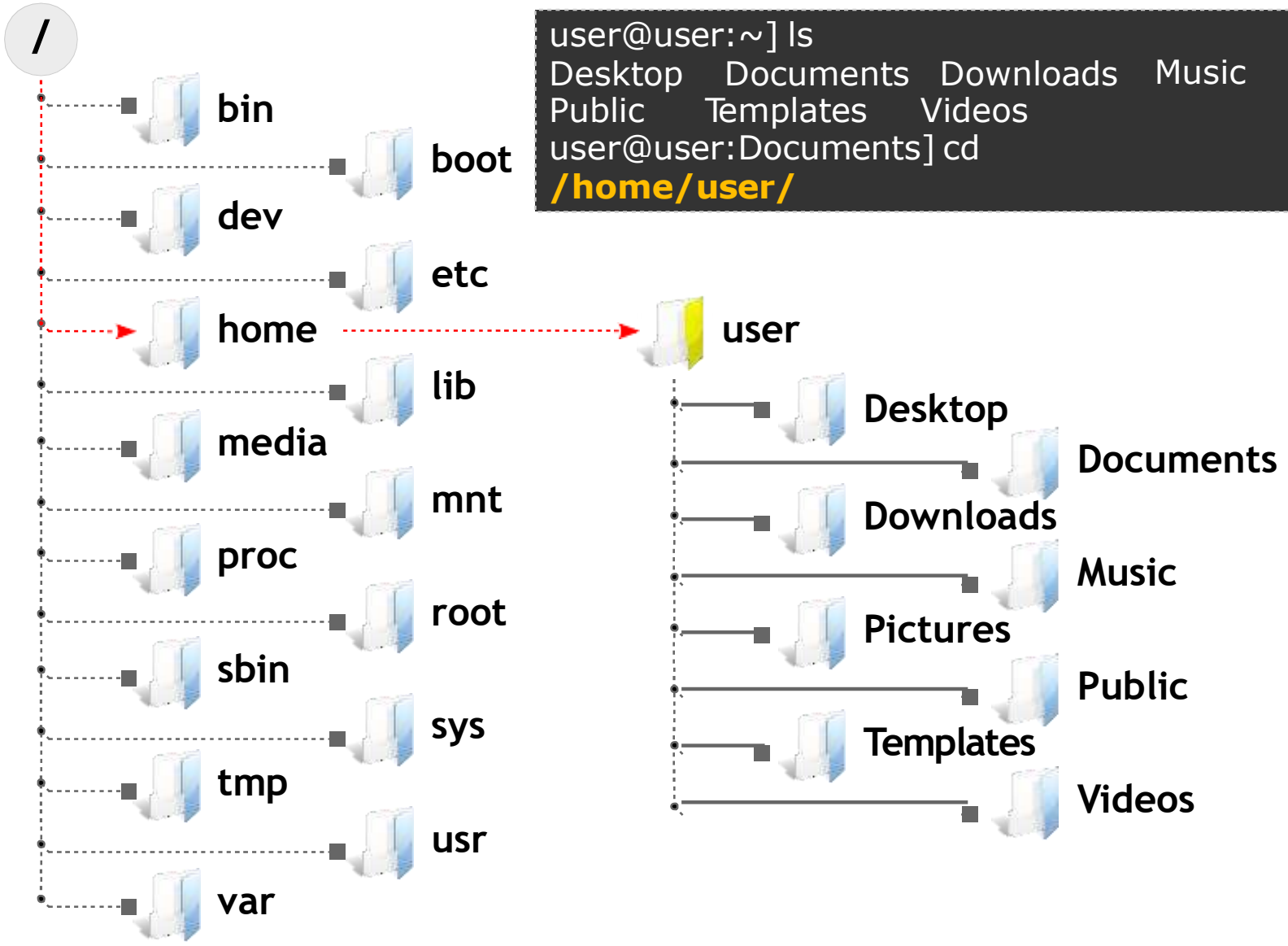
Basic Shell Commands - Path - Relative




```
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures
Public    Templates   Videos
user@user:Documents] cd ..
user@user:~]
```

Linux Systems

Basic Shell Commands - Path -Absoulte




```
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures
Public    Templates   Videos
user@user:Documents] cd
/home/user/
```

Linux Systems

Basic Shell Commands - mkdir

- **mkdir** to create directories

```
user@user:~] ls
Desktop    Documents  Downloads  Music    Pictures     Public     Templates    Videos
user@user:~] mkdir TBD   # TBD - ToBeDeleted
user@user:~] ls
Desktop                Downloads  Music    Pictures     Public     Templates
           Document                TBD
s  Videos
```

- Creates the directory(ies), if they do not already exists

```
user@user:~] ls
Desktop                Downloads  Music    Pictures     Public     Templates
           Document                TBD
suVid@euser:~]mkdir TBD
mkdir: cannot create directory 'TBD': File exists
user@user:~]
```

- Well, # represents start of comment in shell, anything written after that would be seen as comment!

Linux Systems

Basic Shell Commands - rmdir

- **rmdir** to remove empty directories

```
user@user:~] ls
Desktop                 Downloads   Music   Pictures    Public    Templates   TBD
          Document
user@user:~]rmdir TBD
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures    Public    Templates   Videos
user@user:~]
```

- Removes the directory(ies), if they are empty

```
user@user:~] mkdir TBD # TBD - ToBeDeleted
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures    Public    Templates
Videos                                    TBD
user@user:~] cd TBD
user@user:TBD] mkdir Test
user@user:TBD] cd ..
user@user:~] rmdir TBD
rmdir: failed to remove 'TBD': Directory not empty
user@user:~]
```

Linux Systems

Basic Shell Commands - rm

- The previous slide leads to a question on how to del non empty directory?
- **rm** to remove files or directories
- Removes each specified file. By default, it does not remove directories.

```
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures   Public   Templates
Videos                                    TBD
user@user:~] ls TBD
Test
user@user:~] rm TBD
rm: cannot remove 'TBD/': Is a directory
user@user:~]
```

- Then how??, Well we need to refer the **man** pages

Linux Systems

Basic Shell Commands - rm

- From the **man** page or **rm** you find a option **-r** which stands for recursive

```
user@user:~] rm -r TBD
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures   Public   Templates   Videos
user@user:~]
```

- Note, once deleted, you loose the files permanently!, its equivalent to Shift + Delete
- Now what if you delete a file or a folder mistakenly? Wouldn't you like that the shell ask you before you delete? So that you avoid these types of issues!
- Well will see it in the next slide

Linux Systems

Basic Shell Commands - rm

- The **-i** option provides interactivity

```
user@user:~] mkdir TBD # TBD - ToBeDeleted
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures   Public   Templates
Videos                                    TBD
user@user:~] cd TBD
user@user:TBD] mkdir Test
user@user:TBD] cd ..
user@user:~] rm -ri TBD
rm: descend into directory 'TBD/'? y
rm: remove directory 'TBD/Test'? y   rm:
remove directory 'TBD/'? y
user@user:~]
```

- Did you observe from the above screen shot that the shell prompts before you take any action!, you may say **y** or **n**
- There is always a second chance 
- But what if you forget **-i**? 

Linux Systems

Basic Shell Commands - alias

- The solution the question in the previous slide is **ali**
- An builtin bash command, which helps us to name an operation (command), literally anything
- Some built aliases are as shown below

```
user@user:~] alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -l'
alias ls='ls --color=auto'
user@user:~]
```

- List varies based on your system installation
- Well how do we create one? How does it solve our issue?

Linux Systems

Basic Shell Commands - alias

- We may create our own alias with required options a shown below

```
user@user:~] alias rm=’rm -i’
user@user:~] alias
alias egrep='egrep --color=auto'  alias
fgrep='fgrep --color=auto'  alias
grep='grep --color=auto'  alias l='ls -
CF'
alias la='ls -A'
alias ll='ls -l'
alias ls='ls –color=auto'
alias rm=’rm -i’
user@user:~]
```

- Note our alias at the end
- A new command (operation) named **rm** is created, when used henceforth will be **rm -i**

Linux Systems

Basic Shell Commands - alias

- Lets repeat all the steps again to test our new comn

```
user@user:~] mkdir TBD # TBD - ToBeDeleted
user@user:~] ls
Desktop   Documents   Downloads   Music   Pictures   Public   Templates
Videos                                    TBD
user@user:~] cd TBD
user@user:TBD] mkdir Test
user@user:TBD] cd ..
user@user:~] rm -r TBD
rm: descend into directory 'TBD/'? y
rm: remove directory 'TBD/Test'? y   rm:
remove directory 'TBD/'? y
user@user:~]
```

- Observe that, this time we didn't provide the **-i** option!
- How does it work?, its the because of our alias we created.
- The name of new command can be anything, but let it be meaningful

Linux Systems

Basic Shell Commands - alias

- The only issue of the new command is that, it is only available on the terminal it is created??
- As soon as we close the terminal its all gone!!. So what is the point. How do we make it permanent??
- For that we need to about some of the files used by Bash

Linux Systems

Basic Shell Commands - Bash Files

- Hidden files used by the login as well as non-login sh
- Found in users home directory

```
user@user:~] pwd
/home/user
user@user:~] ls -a
.                .bash_logout    .cache    Documents    Pictures    Templates
..               .bash_profile   .config   Downloads    .profile    Videos
.bash_history    .bashrc         Desktop   Music        Public      .vimrc
user@user:~]
```

- These are some configuration files which gets sourced to bash on every new instance is created

Linux Systems

Basic Shell Commands – Bash Files - .bash_profile

- Used by the login shell
- Any action which is to be carried out while logging in through login shell can be put here
- Found in the users home directory, if not could be created

Linux Systems

Basic Shell Commands – Bash Files - .bash_logout

- Any action which is to be carried out while logging c through login shell can be put here
- Found in the users home directory

Linux Systems

Basic Shell Commands – Bash Files - .bashrc

- Used by the non login shell
- Any action which is to be carried after you logged into your machine and open a new terminal window
- This is executed on every new instance of the new terminal even if it is in a form of tabs
- So all those the stuff you would like to get sourced the bash while starting, then this would be right file
- By adding the alias in this file would make it persistent which is raised a question in this slide

Linux Systems

Basic Shell Commands – Bash Files - .bash_history

- A file which store all the user activity on the comma prompt for future reference.
- The **history** command shows the entries of this file

Linux Systems

Basic Shell Commands - touch

- An command which helps us to update the timestamp of the existing files
- If the file argument that does not exist is created empty

```
user@user:~] mkdir TBD # TBD - ToBeDeleted
user@user:~] cd TBD
user@user:TBD] ls
user@user:TBD] touch file1 file2 file3
user@user:TBD] ls
file1    file2    file3
user@user:TBD]
```

- Observe that the directory was empty, using touch command it created the mentioned files
- As mentioned it is generally used to update the time stamp, so you wonder what would be the use of it?
- Well, they are used by some advanced commands like **make**

Linux Systems

Basic Shell Commands - touch

- The scope of this topic is to create some empty files which will help learning some of the commands we are going cover next slides
- But lets see the behavior of the command a bit

```
user@user:TBD] ls -l
total 0
-rw-rw-r-- 1 user user 0 Nov 30 22:18 file1
-rw-rw-r-- 1 user user 0 Nov 30 22:18 file2
-rw-rw-r-- 1 user user 0 Nov 30 22:18 file3
user@user:TBD]
```

- Note the creation time of all the files

```
user@user:TBD] touch file1 file2 file3 # 20 Seconds after creation!
user@user:TBD] ls -l
total 0
-rw-rw-r-- 1 user user 0 Nov 30 22:38 file1
-rw-rw-r-- 1 user user 0 Nov 30 22:38 file2
-rw-rw-r-- 1 user user 0 Nov 30 22:38 file3
user@user:TBD]
```

Linux Systems

Basic Shell Commands - cp

- **cp** to copy files and directory
- Accepts to 2 arguments as the source and destiny

```
user@user:TBD] cp <source> <destiny>
```

- The source and destiny are the path of files to be copied from and to

```
user@user:TBD] ls
file1    file2    file3
user@user:TBD] cp file1 file4
user@user:TBD] ls
file1    file2    file3    file4
user@user:TBD] cp file1 ../file5
user@user:TBD] ls ../
Desktop   Documents   Downloads   file5   Music   Pictures   Public   Templates
TBD       Videos
user@user:TBD] cp ../file5 .
user@user:TBD] ls
file1    file2    file3    file4    file5
user@user:TBD]
```

Linux Systems

Basic Shell Commands - cp

- While copying the directory the **-r** option has to be used

```
user@user:TBD] mkdir Test
user@user:TBD] ls
file1      file2      file3      file4      file5      Test
user@user:TBD] cp Test/ TestCopy
cp: -r not specified; omitting directory 'Test/'
user@user:TBD] cp -r Test/ TestCopy   user@user:TBD]
ls
file1      file2      file3      file4      file5      Test
           TestCopy  user@user:TBD] cp file1 file2 file3
Test   user@user:TBD] ls Test/
file1      file2      file3
user@user:TBD]
```

Linux Systems

Basic Shell Commands - mv

- **mv** to move of rename files and directories
- Accepts to 2 arguments as the source and destiny

```
user@user:TBD] mv <source> <destiny>
```

- The source and destiny are the path of files to be moved or renamed from and to

```
user@user:TBD] ls ../
Desktop   Documents   Downloads   file5   Music   Pictures   Public   Templates
TBD       Videos
user@user:TBD] mv ../file5 TestCopy
user@user:TBD] ls TestCopy
file5
user@user:TBD] ls ../
Desktop   Documents   Downloads   Music   Pictures   Public   Templates
Videos                            TBD
user@user:TBD]
```

- Note the above context, the file gets moved

Linux Systems

Basic Shell Commands - mv

```
user@user:TBD] ls
file1     file2     file3     file4     file5     Test     TestCopy
user@user:TBD] mv file1 file1.txt
user@user:TBD] ls
file1.txt     file2     file3     file4     file5     Test     TestCopy
user@user:TBD]
```

- In this context, the file gets renamed

Visual Editor - vi

Linux Systems

Visual Editor - vi

- Screen-oriented text editor originally created for the Unix operating system
- The name vi is derived from the shortest unambiguous abbreviation for the ex command visual
- Improved version is called as vim
- To open a file

```
user@user:TBD] vi <file_name>
```

or

```
user@user:TBD] vim <file_name>
```

Linux Systems

Visual Editor - vim

- **vi** opens a file in command mode to start mode.
- The power of **vi** comes from the following modes

Mode	Functions	Key
Escape	Search File Edit	Esc
Edit	Insert Replace	i / I r / R
Visual	Selection	v / V
Command Line	Commands	:

- To get a basic grip on **vi** you may type **vimtutor** on the terminal. Just read and follow the instructions

```
user@user:TBD] vimtutor
```

Linux Systems

Visual Editor – vim - Information

```
user@user:TBD] ls
file1.txt       file2     file3     file4     file5     Test     TestCopy
user@user:TBD] vi file.txt
```

filename,
mode,
previous action
command prompt

file
command
information

line, cursor

view %

Linux Systems

Visual Editor – vim - Modes

```
This mode is default as soon as you open a file. You may press ESC key any   time to
enter this mod
~
                                                                2,23          All
```

Escape Mode. Press ESC

```
This is the mode where, you would be entering the text. Most of the file   Commands
will not work here!. Press i anytime while in ESC mode
~
-- INSERT --                                                    2,63          All
```

Insert Mode. Press i

```
      The existing text will        be over written or replaced     with new one. You may
    press R anytime to get          into this mode while in ESC  mode
~
-- REPLACE --                                                   2,55          All
```

Replace Mode. Press ESC Press R

Linux Systems

Visual Editor - vim - Modes

```
This mode helps us to select a part of the content.
You may press ESC and v to enter into this mode and your arrow keys to select the text
you want t you may use any of the file edit command and perform the required
opertion. Observe the selection made above!!
~
-- VISUAL --                                    2        3,18          All
```

Visual Mode. Press ESC Press v

```
This mode helps us to ct a part of the
You may press ESC and TRL v to enter into node and your arrow keys to select the
block you w you file edit command and perform th
                Observe the selection made!!
~
-- VISUAL BLOCK --                              4x20     4,42          All
```

Visual Block Mode. Press ESC Press Ctrl v

Linux Systems

Visual Editor – vim - Modes

```
Press ESC and : brings us in command mode.
You may use the command required and press ENTER
~
~
:<Type your command here>
```

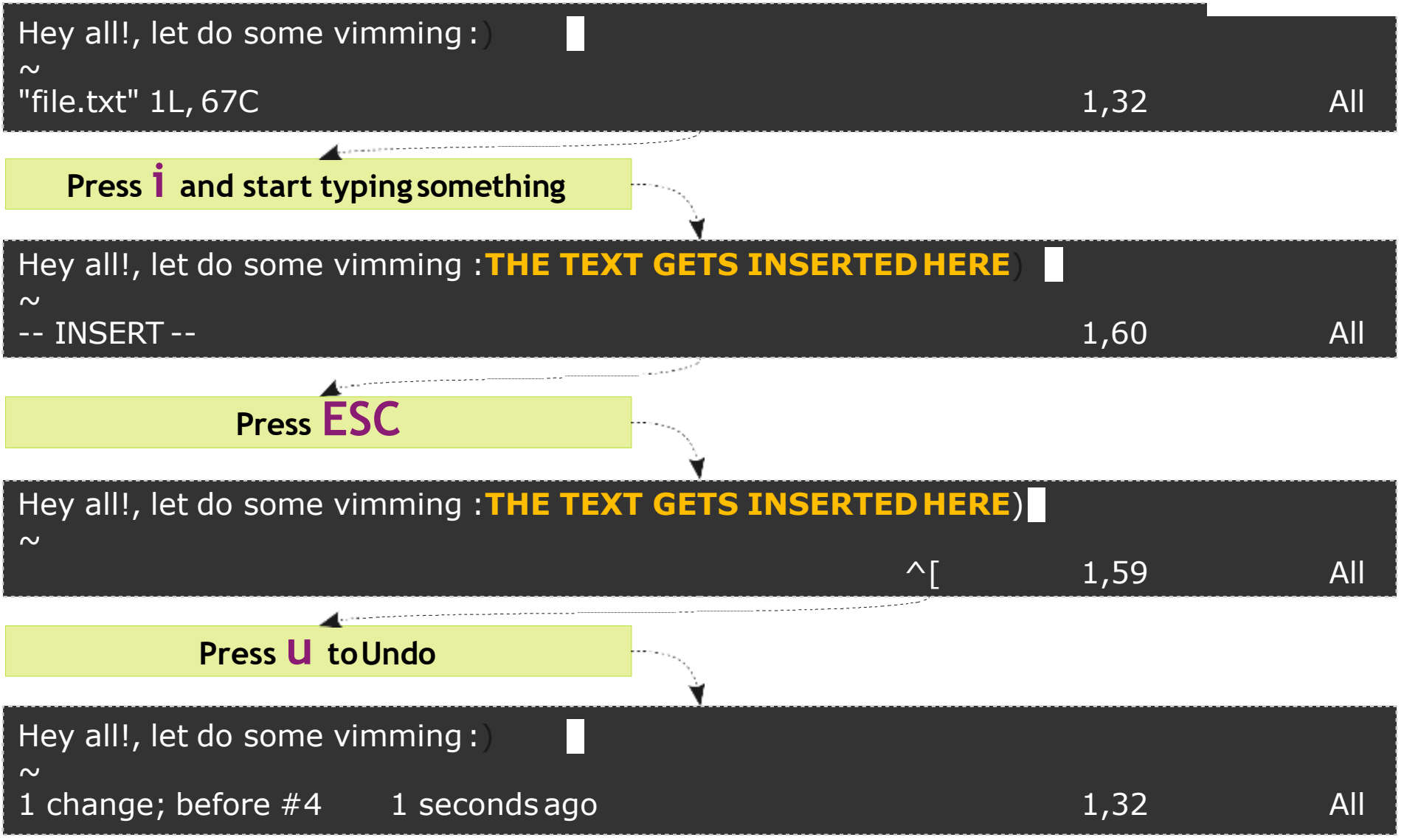
Command Line Mode. Press ESC Press:

Linux Systems

Visual Editor - vi - Insert

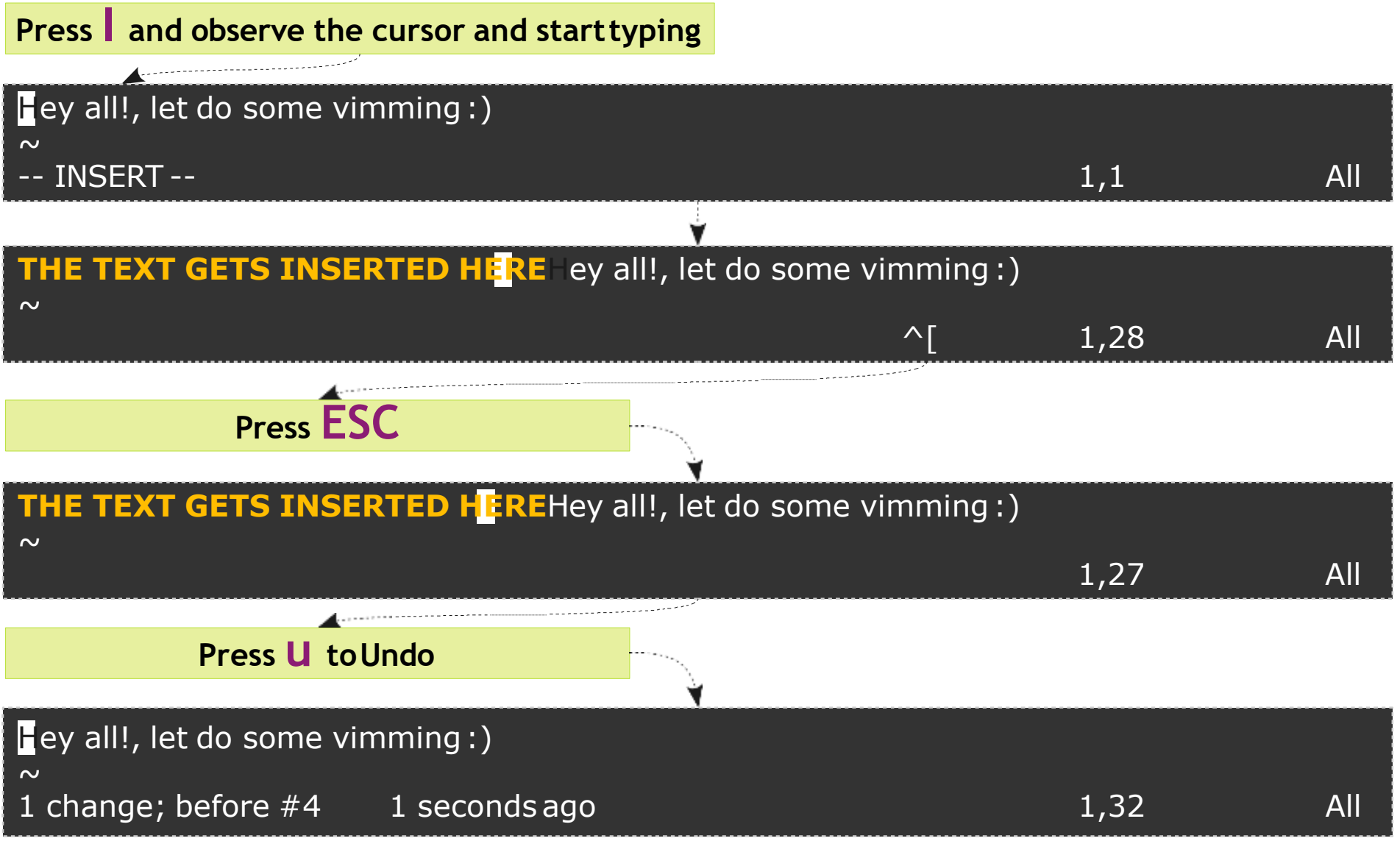
Linux Systems

Visual Editor - vi - Insert and Undo



Linux Systems

Visual Editor - vi - Insert



Linux Systems

Visual Editor - vi - Write and Quit

```
Hey all!, let do some vimming :)
~
~
~
:w
```

To save. Press **w** and **ENTER**

```
Hey all!, let do some vimming :)
~
~
~
:"test.txt" 1L, 57C [w]                                  1,56          All
```

```
Hey all!, let do some vimming :)
~
~
~
:q
```

To quit. Press **q** and **ENTER**

Linux Systems

Visual Editor – vi – Write and Quit

```
user@user:TBD] ls
file1.txt       file2      file3      file4      file5      Test      TestCopy
user@user:TBD] vi test.txt
user@user:TBD] ls
file1.txt       file2      file3      file4      file5      Test      TestCopy      test.txt
user@user:TBD]
```

- Just to understand this better, lets learn the **cat** command before we proceed further

Linux Systems

vi - Basic Shell Commands - cat

- **cat** to concatenate files and print on standard output

```
user@user:TBD] ls
file1.txt       file2     file3     file4     file5     Test     TestCopy
user@user:TBD] vi test.txt
user@user:TBD] ls  file1.txt
                file2     file3     file4     file5     Test     TestCopy     test.txt
user@user:TBD] cat test.txt
Hey all!, let do some vimming :)
user@user:TBD]
```

- Displays the contents of the file on the screen
- **cat** has many advanced functionalities which will be covered later as required

Linux Systems

Visual Editor - vi - Write and Quit

```
user@user:TBD] vi test.txt
```

```
Hey all!, let do some vimming :)
~
~
~
:wq
```

To save and quit. Press **wq** and **ENTER**

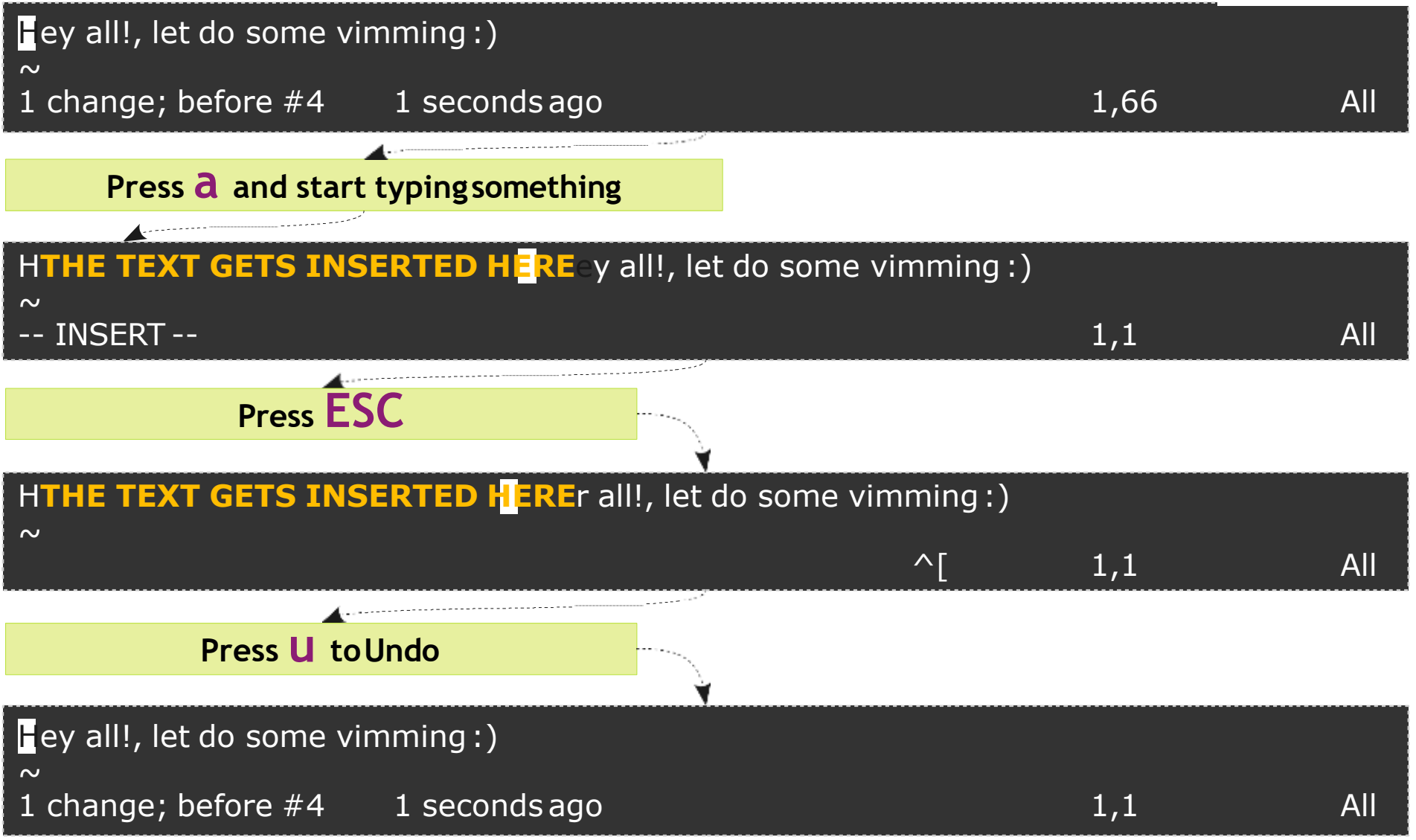
```
user@user:TBD] vi test.txt
```

```
Hey all!, let do some vimming :)
Added this line too!! which will be missing!
~
~
:q!
```

To quit without saving. Press **q!** and **ENTER**

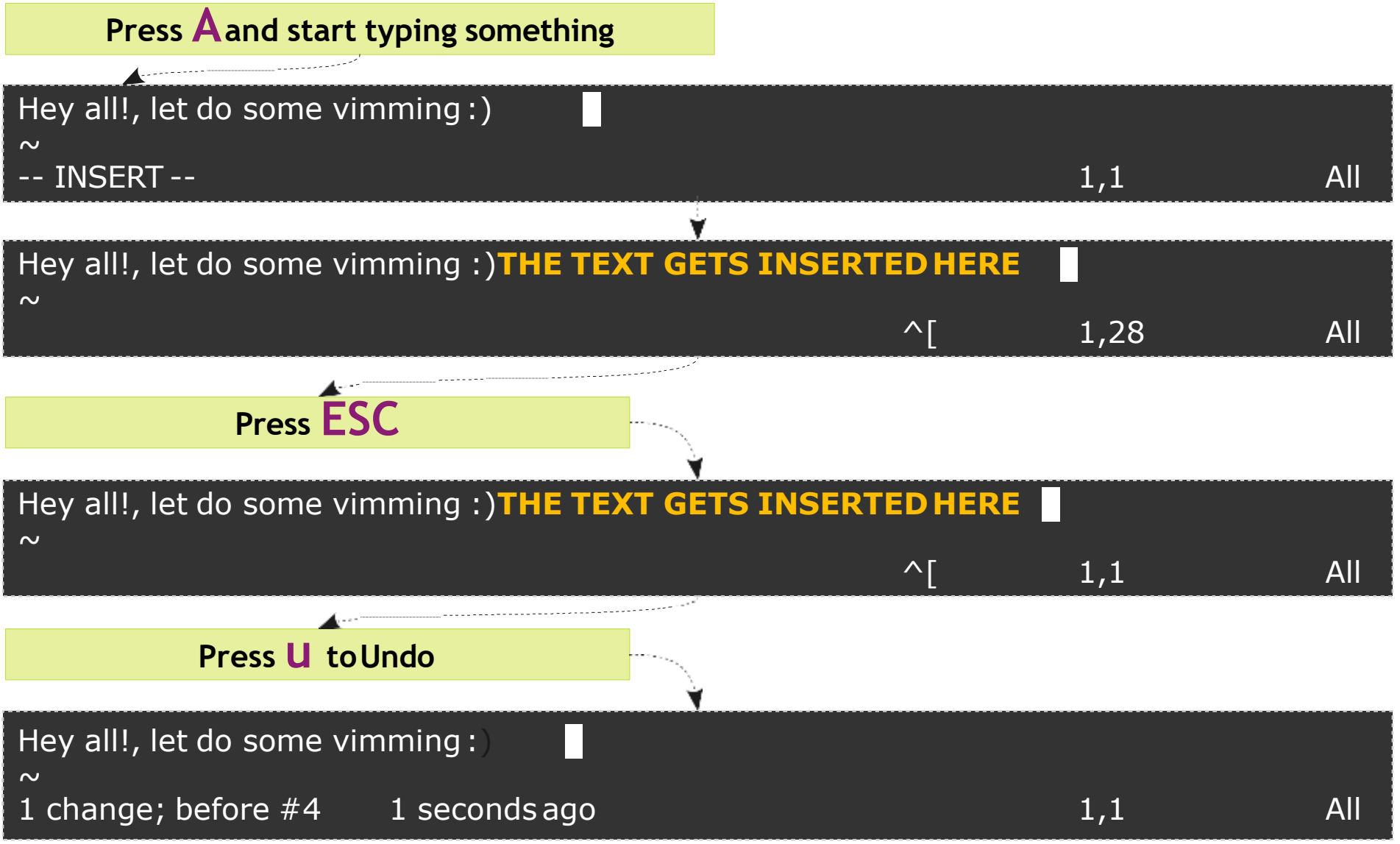
Linux Systems

Visual Editor - vi - Append



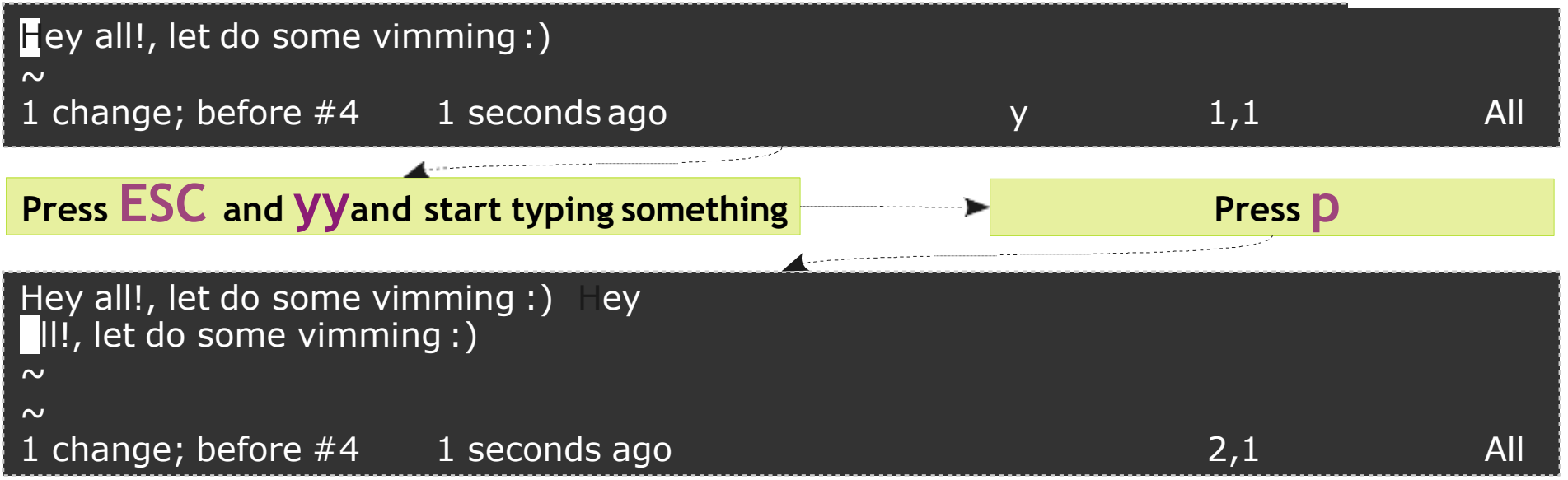
Linux Systems

Visual Editor - vi - Append



Linux Systems

Visual Editor – vi – Yank (Copy) and Paste



- You may copy multiple lines and paste times using the following syntax
  - [n]yy - Copy n line(s) → Default n = 10, 10yy copies 10 lines
  - [n]p - Paste copied line(s) n time → Default n = 10, 10p pastes the copied line(s) 10 times
- Note, this rule applies to almost all the commands

Linux Systems

Visual Editor - vi - Open

Linux Systems

Visual Editor - vi - Open

Linux Systems

Visual Editor - vi - Increment

```
1 line Hey all!, let do some vimming :)
~
1 change; before #4       1 seconds ago                    y              1,1              All
```

Press ESC and yy and start typing something → Press p

```
1 line Hey all!, let do some vimming :)
1 line Hey all!, let do some vimming :)
~
~
2 fewer lines;        #4   1 seconds ago                              2,1              All
```

Press CRTL a and observe the 2nd line

```
1  line Hey all!, let do some vimming :)
2  line Hey all!, let do some vimming :)
~
~
2 fewer lines;        #4   1 seconds ago                              2,1              All
```

Press yyp

Linux Systems

Visual Editor - vi - Go

```
1  line Hey all!, let do some vimming :)
2line Hey all!, let do some vimming :)
2 line Hey all!, let do some vimming :)
~
2 fewer lines;        #4   1 seconds ago                    3,1            All
```

Press CRTL a and observe the 3rd line

```
1  line Hey all!, let do some vimming :)
2  line Hey all!, let do some vimming :)
3  line Hey all!, let do some vimming :)
~
2 fewer lines;        #4   1 seconds ago                    3,1            All
```

Press gg and observe the cursorposition

```
1  line Hey all!, let do some vimming :)
2  line Hey all!, let do some vimming :)
3  line Hey all!, let do some vimming :)
~
2 fewer lines;        #4   1 seconds ago                    1,1            All
```

Linux Systems

Visual Editor – vi – Go

- The most useful shortcut to navigate between different lines
  - [n]gg - Go to nth line → Default n = 10, 10gg takes you to 10 line
  - gg take you the first line of the file
  - G take you the last line of the file

Linux Systems

Visual Editor - vi - Delete

```
1 line Hey all!, let do some vimming :)
2 line Hey all!, let do some vimming :)
3 line Hey all!, let do some vimming :)
~
2 fewer lines;       #4   1 seconds ago                    1,1           All
```

Press **2dd** and observe

```
3 line Hey all!, let do some vimming :)
~
~
~
2 fewer lines;       #4   1 seconds ago                    1,1           All
```

- You may delete multiple lines from the current cursor position
  - [n]**dd** - Delete n line(s) → Default n = 10, 10dd delete 10 lines
  - **D** the current line from the cursor position

Linux Systems

Visual Editor - vi - Decrement

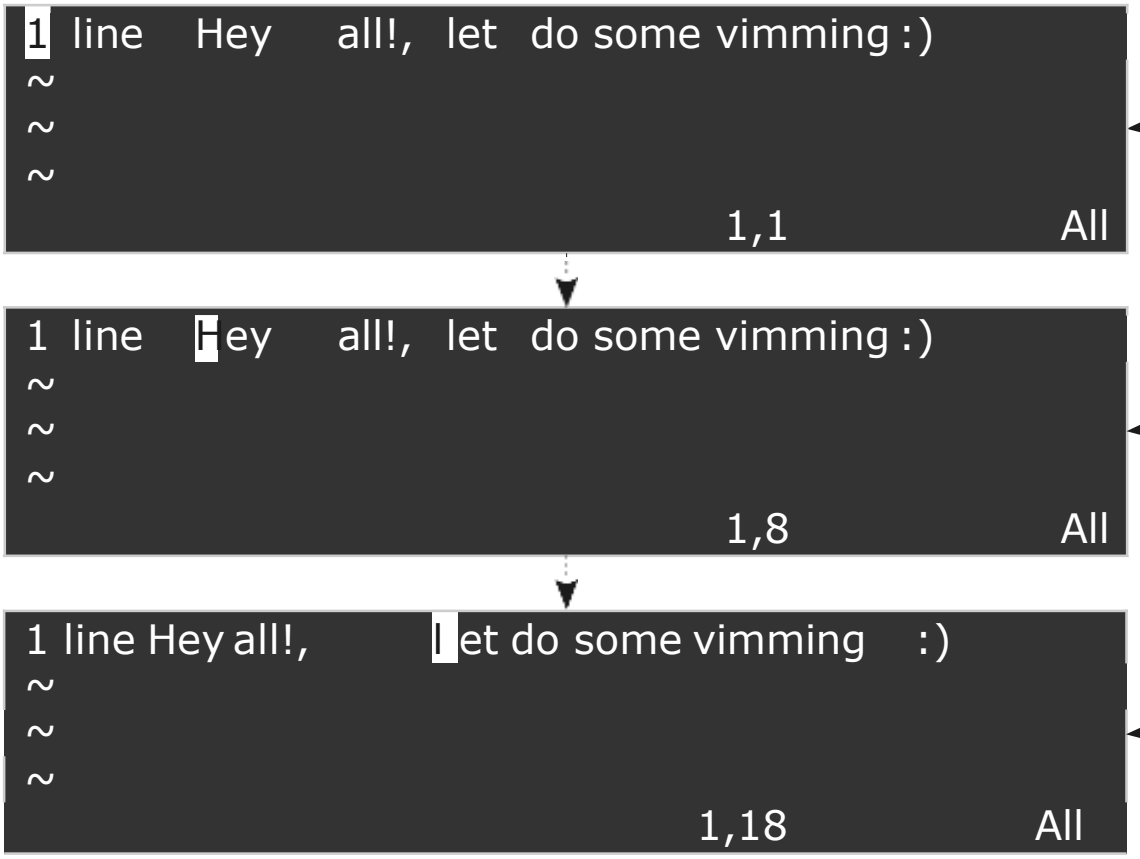
Press 2 CTRL x and observe

```
1 line Hey all!, let do some vimming :)
~
~
~
2 fewer lines;        #4   1 seconds ago                                  1,1                All
```

- You may increment and decrement n times
  - [n]CTRL x - Increments the first integer match from the cursor position → Default n = 10 and number is 13, 10 CTRL x decrements the number by 10, resulting to number 3
  - The increments works the same way

Linux Systems

Visual Editor – vi – Navigation - Forward



Press W and observe cursor

Press W and observe cursor

Press 3W and observe cursor

- [n]w – Move forward n words → Delay n = 10w – move 10 words ahead

Linux Systems

Visual Editor – vi – Navigation - Backward

Press **b** and observe cursor

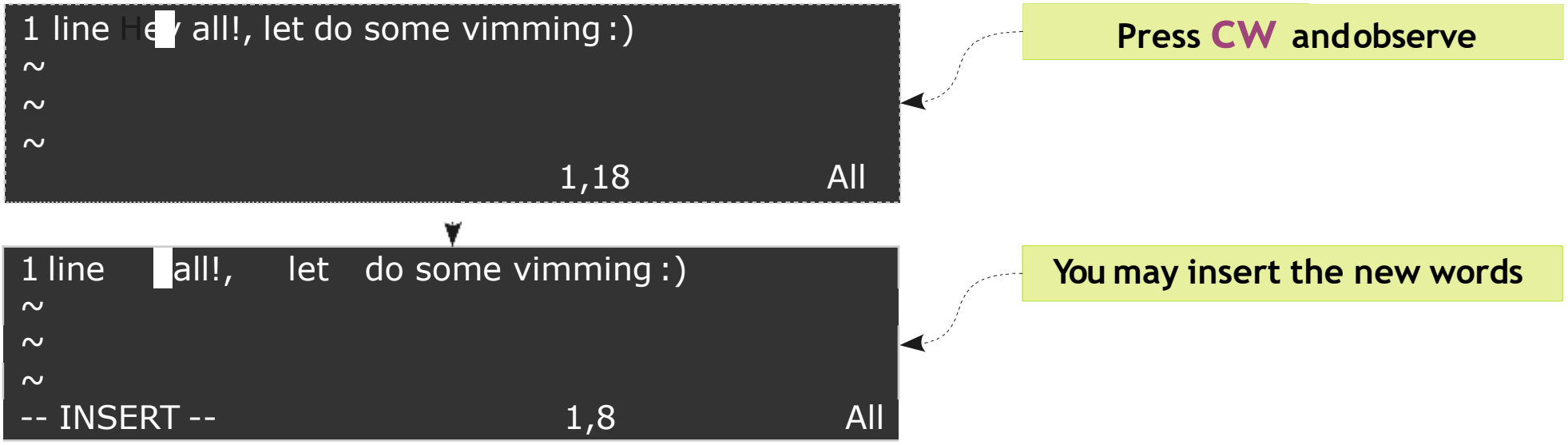
Press **b** and observe cursor

Press **2b** and observe cursor

- [n]**b** – Move backward n words → Delay n = 10b – move 10 words behind

Linux Systems

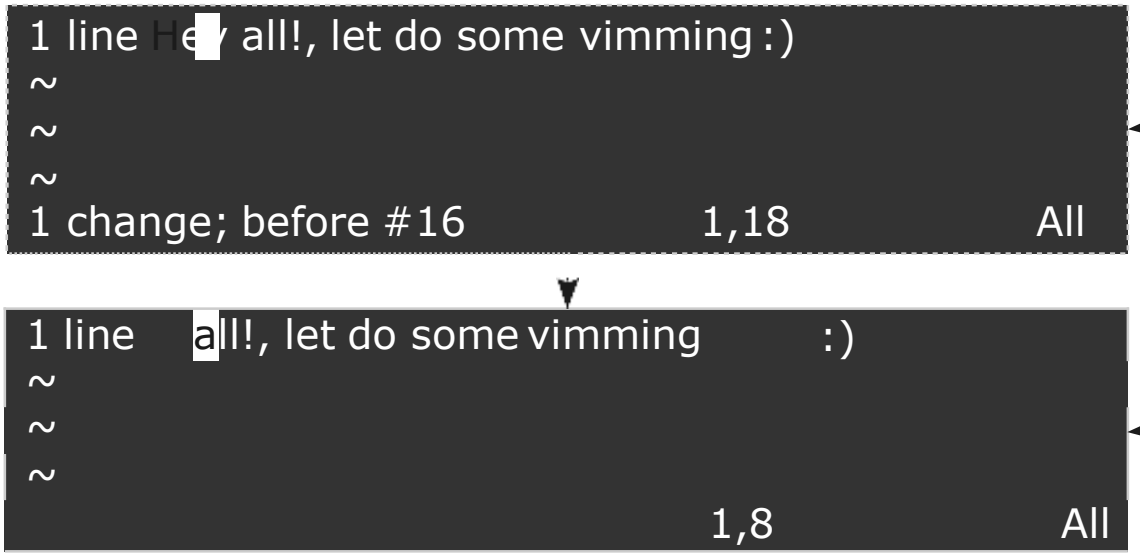
Visual Editor – vi – Change Word



- [n]cw – Change n words – Delete n = 10cw - change 10 words from the current cursor position

Linux Systems

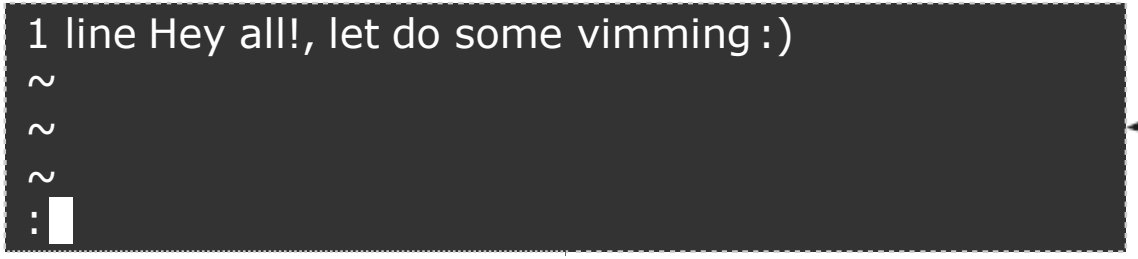
Visual Editor - vi - Delete Word



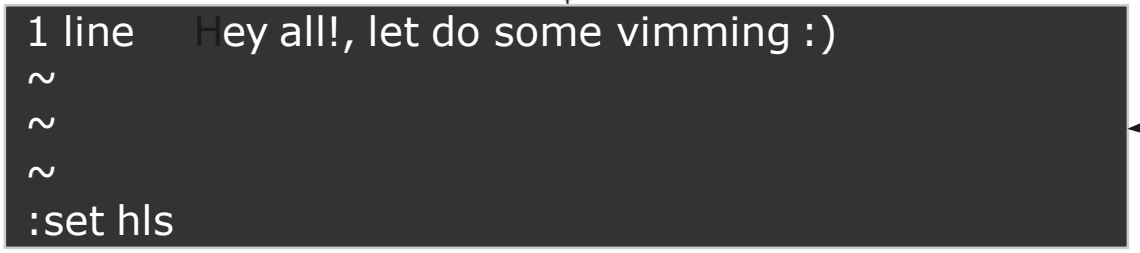
- [n]dw - Delete n words → Delete n = 10dw - delete 10 words from the current cursor position

Linux Systems

Visual Editor - vi - Settings



Press : to enter Command Line Mode

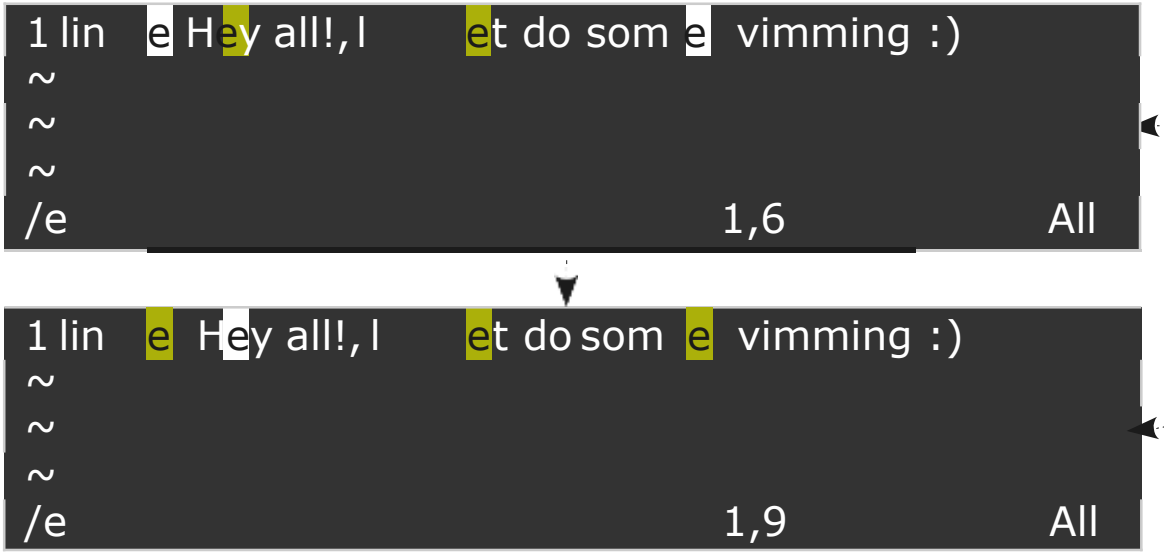


Type set hls and ENTER to enable search highlights

- :set hls - Enable highlight search
- :set nohls - Disable highlight search
- :set nu - Enable line numbers
- :set nonu - Disable line numbers

Linux Systems

Visual Editor – vi – Search - Forward



- [n]n – Forward search n words → Let's say n = 10n – Searches the 10 words from the current position forward

Linux Systems

Visual Editor – vi – Search - Backward

Press N

Press 3N and observe the cursor position

- [n]n – Reverse search n words – Default n = 10n - Searches the 10 words from the current position backwards

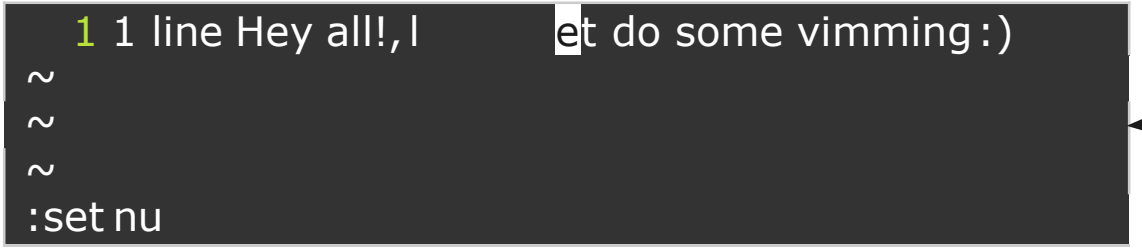
Type :set nohls and ENTER to disable search highlights

Linux Systems

Visual Editor - vi - Settings

Press : to enter CommandLine Mode

Type set hls andENTER to enable search highlights

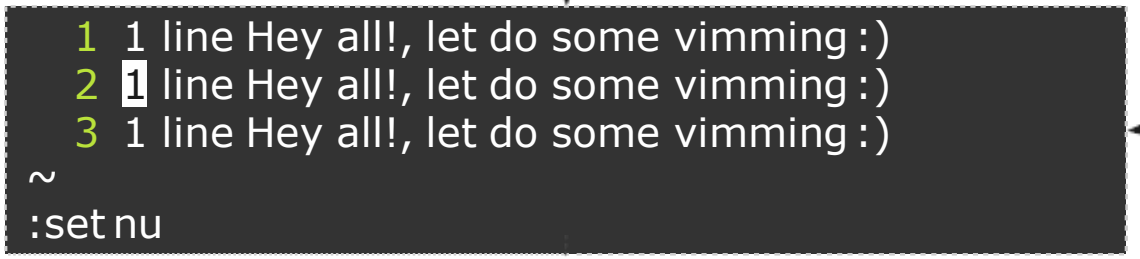


Type set nu andENTER to enable line numbers

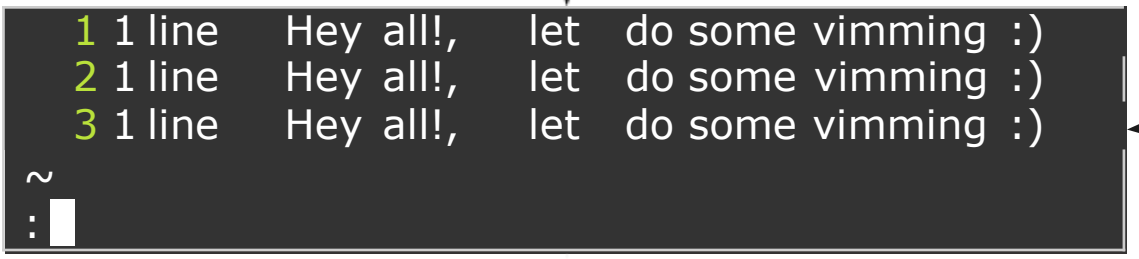
Linux Systems

Visual Editor - vi - Substitute

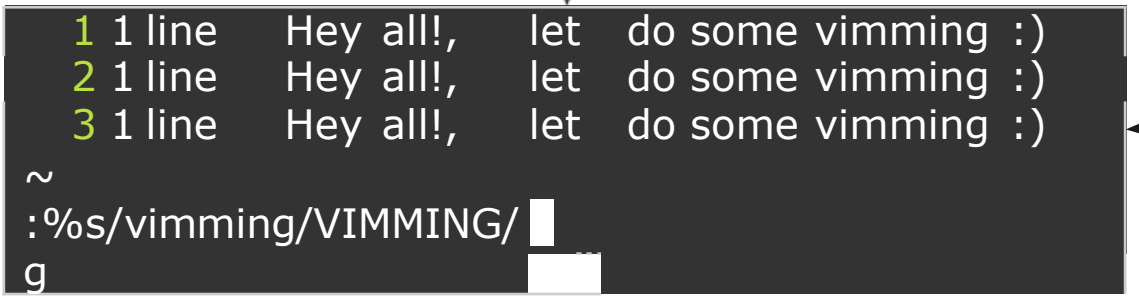
Type yy and 2p



Press : to enter CommandLine Mode



Press : to enter CommandLine Mode



Type %s/vimming/VIMMING/g

Linux Systems

Visual Editor - vi - Substitute

```
  1  1 line Hey all!, let do some VIMMING :)
  2  1 line Hey all!, let do some VIMMING :)
  3    line Hey all!, let do some VIMMING :)
~
3 substitutions on 3 lines
```

Substitutes the pattern globally

```
  1  1 line Hey all!, let do some VIMMING :)
  2  1 line Hey all!, let do some VIMMING :)
  3    line HEY all!, let do some VIMMING :)
~
:3s/Hey/HEY/g
```

Substitutes the pattern on the Specified line(s)

```
  1  1 linEE Hey all!, let do some VIMMING :)
  2  1 linEE Hey all!, let do some VIMMING :)
  3    linEE HEY all!, let do some VIMMING :)
~
:%s/e/EE/
```

Substitutes the first occurrence of pattern globally

- Interesting pattern substitutions possible if have knowledge on Regular Expressions

Linux Systems

Visual Editor – vi – Edit and Read

- Edit
  - :e filename - open another file without closing the current
  - To switch between these files use CTRL 6. Make sure the file is save
  - You will not be able to undo after the switch
- Read
  - :r filename - reads file named filename at the current cursor position

Shell Scripting – Part 1

Linux Systems

Shell Scripting - Programming Languages

- There are various types of programming languages, compared on various parameters
- From Embedded system engineer's view it should be seen how close or how much away from the hardware the language is
- Based on that view programming languages can be categorized into three areas:
  - Assembly language (ex: 8051)
  - Middle level language (ex: C)
  - High level / Scripting language (ex: Shell)

Linux Systems

Shell Scripting - Programming Languages

- Each programming language offers some benefits with some shortcomings
- Depending on the need of the situation appropriate language needs to be chosen
- This make language selection is a key criteria when it comes to building real time products!

Linux Systems

Shell Scripting – Prog... Lang... – A Comparison

Language parameter	Assembly	C	Shell
Speed	High	Medium	Medium
Portability	Low	Medium	High
Maintainability	Low	Medium	High
Size	Low	Medium	Low
Easy to learn	Low	Medium	High

Shell or any scripting language is also called as ‘interpreted’ language as it doesn’t go through compilation phase. This is to keep the language simple as the purpose is different than other languages.

Linux Systems

Shell Scripting – What is a Script?

- Any collection of shell commands can be stored in a file, which is then called as shell script
- Programming the scripts is called shell scripting
- Scripts have variables and flow control statements like other programming languages
- Shell script are interpreted, not compiled
- The shell reads commands from the script line by line and searches for those commands on the system

Linux Systems

Shell Scripting - Script – Where to use?

- System Administration
  - Automate tasks
  - Repeated tasks
- Development
  - Allows testing a limited sub-set
  - Testing tools
- Daily usage
  - Simple scripts
  - Reminders, e-mails etc…

Linux Systems

Shell Scripting - Script – Example

```
user@user:~] cd # Move to home directory
user@user:~] mkdir -p ECEP/LinuxSystems/Classwork
user@user:~] cd ECEP/LinuxSystems/Classwork
user@user:Classwork]
```

```
user@user:Classwork] vi hello.sh
```

```
 1  #!/bin/bash
 2  echo "Hello World"            # Print Hello World on standard output
~
~
:wq
```

```
user@user:Classwork] bash hello.sh
Hello World
user@user:Classwork] ls -l
-rw-rw-r-- 1 adil adil 31 Dec  2 21:44 hello.sh
user@user:Classwork] chmod +x hello.sh
user@user:Classwork] ls -l
-rwxrwxr-x  1 adil adil  31 Dec  2 21:44 hello.sh
user@user:Classwork] ./hello.sh
Hello World
user@user:Classwork]
```

Linux Systems

Shell Scripting - Script – echo

- **echo** displays a line of text

```
user@user:Classwork] echo Hello World
Hello World
user@user:Classwork]
```

- Helps use to print on screen with required formatting
- Used in scripts to print normal messages and the value of variable, which will be seeing shortly
- Some examples are as shown below

```
user@user:Classwork]   echo   Hello\nWorld # To   print on 2 seperate line
Hello\nWorld
user@user:Classwork]   echo    -e "Hello\nWorld"    # To print on 2 seperate line
Hello
World
user@user:Classwork]   echo    -e "Hello\tWorld"    # To words with tab
Hello     World
user@user:Classwork]   echo    -e "Hello\rWorld"    # To overwrite previous print
World
user@user:Classwork]
```

Linux Systems

Shell Scripting - Special Characters

- Characters which has special meaning
- Used in many advanced functionalities
- Lets understand some of the most important ones

~ → Denotes the current user's home directory

```
user@user:Classwork] echo ~ # Expands to users home directory path
/home/user
user@user:Classwork]
```

& → Denotes applications or commands in the background

```
user@user:Classwork] firefox
      ⇐ Observe here, the shell would be held by the application you invoked!!
```



```
user@user:Classwork] firefox & # firefox is opened in background [1] 7746
user@user:Classwork] # You may use the command prompt
```

Linux Systems

Shell Scripting - Special Characters

* →wildcard, matching zero or more characters (e.g. : ls doc_*)

```
user@user:Classwork] mkdir TBD
user@user:TBD] cd TBD
user@user:TBD] touch file_{1..5}.txt # Create 5 files from 1 to 5
user@user:TBD] touch file_{6..10}.c
user@user:TBD] touch file_{1..5}.sh
user@user:TBD] ls
file_10.c       file_13.sh      file_1.txt      file_4.txt      file_7.c
file_11.sh      file_14.sh      file_2.txt      file_5.txt      file_8.c
file_12.sh      file_15.sh      file_3.txt      file_6.c        file_9.c
user@user:TBD] ls *.txt
file_1.txt      file_2.txt      file_3.txt      file_4.txt      file_5.txt
user@user:TBD] ls *1*
file_10.c       file_12.sh      file_14.sh      file_1.txt
file_11.sh      file_13.sh      file_15.sh
user@user:TBD]
```

Linux Systems

Shell Scripting - Special Characters

? →wildcard, matching exactly one character (e.g.: ls doc_?)

```
user@user:TBD] ls
file_10.c       file_13.sh      file_1.txt      file_4.txt          file_7.c
file_11.sh      file_14.sh      file_2.txt      file_5.txt          file_8.c
file_12.sh      file_15.sh      file_3.txt      file_6.c  file_9.c
user@user:TBD] ls file_?.c
file_6.c      file_7.c      file_8.c      file_9.c
user@user:TBD] ls file_??.c
file_10.c
user@user:TBD] ls file_??.*
file_10.c      file_11.sh      file_12.sh      file_13.sh      file_14.sh      file_15.sh
user@user:TBD]
```

Linux Systems

Shell Scripting - Special Characters

$ →used to access a variable (e.g. : $HOME), used with **echo** command

```
user@user:TBD]  echo  $ # Note, A $ with any non space character  see     will be seen
as a variable.      Will    this in next few slides
$
user@user:TBD] echo $0 # Expands to name of the shell or shellscript
/bin/bash
user@user:TBD] echo $$ # Expands to this shell process id
2668
user@user:TBD] echo $? # Expands to the status of the previous command
0
user@user:TBD]
```

- There 2 more which makes sense in a script will see them shortly
  - $@ →Deref value of all arguments passed
  - $# →Deref no of arguments passed to shell script

Linux Systems

Shell Scripting - Variables

- Variables are a way of storing information temporarily

A couple of conversions we need to follow

- Variables usually appear in uppercase
- There should not be a white space between the variable

```
user@user:TBD] X=10
user@user:TBD] echo $X # Any non space charater adjacent to $ is a variable
10
user@user:TBD] NAME=”EMERTXE”
user@user:TBD] echo $NAME
EMERTXE
user@user:TBD] echo $DUMMY # The variable is not set. Hence no print
DUMMY
user@user:TBD]
```

Linux Systems

Shell Scripting - White-space & Line-breaks

- Bash shell scripts are very sensitive to white-space & line- breaks
- Because the “keywords” of this programming language are actually commands evaluated by the shell
- Need to separate arguments with white spaces
- Likewise a line-break in the middle of a command will mislead the shell into thinking the command is incomplete.

```
user@user:TBD] TEST = 10 # Not allowed
TEST: command not found
user@user:TBD] TEST1=10; TEST2=20 # Allowed
user@user:TBD] echo $TEST1 $TEST2
10 20
user@user:TBD]
```

Linux Systems

Shell Scripting - The Shell Env Variables

- Login-shell's responsibility is to set the non-login shell and it will set the environment variables
- Environment variables are set for every shell and generally at login time
- Environmental variables are set by the system.
- env - lists shell environment variable/value pairs

Linux Systems

Shell Scripting - The Shell Env Variables

```
user@user:TBD] echo $HOME # The current user's home directory
/home/user
user@user:TBD] echo $SHELL # Shell that will be interpreting user commands
/bin/bash
user@user:TBD] echo $USER # The current logged in user
user
user@user:TBD] echo $PWD # The previous working directory
/home/user/ECEP/LinuxSystems/Classwork/TBD
user@user:TBD] echo $OLDPWD # The previous working directory
/home/user
user@user:TBD] echo $PATH # System will check when looking for commands here
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/
local/games:/snap/bin:
user@user:TBD] echo $HOSTNAME # The hostname of the computer
emertxe
user@user:TBD] echo $TERM # Type of terminal to emulate when running the shell
xterm-256color
user@user:TBD] echo $PS1
\[\e]0;\u@\h:
\w\a\]${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\
033[00m\]:\[\033[01;34m\]\W\[\033[00m\]]
user@user:TBD]
```

Linux Systems

Shell Scripting - Quotes

- Using double quotes to show a string of characters will allow any variables in the quotes to be resolved

```
user@user:TBD] VAR=10
user@user:TBD] echo $VAR
10
user@user:TBD] echo “The value is $VAR”
The value is 10
user@user:TBD]
```

- Using single quotes causes the variable name to be used literally, and no substitution will take place

```
user@user:TBD] VAR=10
user@user:TBD] echo $VAR
10
user@user:TBD] echo ‘The value is $VAR’
The value is $VAR
user@user:TBD]
```

Linux Systems

Shell Scripting - Expressions

- **expr** Evaluates simple math on the command line calculator

```
user@user:TBD] expr  10 + 20
30
user@user:TBD] expr  10 * 20
expr: syntax error
user@user:TBD] expr  10 \* 20 # * acts a wildcard, so need to escape          it
200
user@user:TBD]
```

- **bc** An arbitrary precision calculator language

```
user@user:TBD] bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundati
on, Inc. This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'./
10 * 20
200
10 + 34 * 23
792
quit
user@user:TBD]
```

Linux Systems

Shell Scripting – Expressions - Script

```
user@user:TBD] cd ../PPT-Examples
user@user:PPT-Examples]
```

```
user@user:PPT-Examples] vi 1_expression.sh
```

```
  1 #!/bin/bash
  2
  3 NUM1=5
  4 NUM2=3
  5
  6 ADD=$((${NUM1} + ${NUM2}))
  7 SUB=$((${NUM1} - ${NUM2}))
  8 MUL=$((${NUM1} * ${NUM2}))
  9 DIV=$((${NUM1} / ${NUM2}))
 10 MOD=$((${NUM1} %
                ${NUM2}))
 11
 12 echo -e "Addition of two numbers is\t\t: ${ADD}"
 13 echo -e "Substraction of two numbers is\t\t: ${SUB}"
 14 echo -e "Multiplication of two numbers is\t: ${MUL}"
 15 echo -e "Division of two numbers is\t\t: ${DIV}"
 16 echo -e "Modulum of two numbers is\t\t: ${MOD}"
~"1_expression.sh" 16L, 418C                    0,0-1          All
```

Linux Systems

Shell Scripting – Expressions - Script

```
user@user:PPT-Examples] chmod +x 1_expression.sh
user@user:PPT-Examples] ./1_expression.sh  Addition
of two numbers is                          : 8
Substraction of two numbers is             : 2
Multiplication of two numbers is           : 15
Division of two numbers is                 : 1
Modulum of two numbers is                  : 2
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting - Conditions – if else

- The if statement chooses between alternatives each of which may have a complex test
- The simplest form is the if-then statement

Syntax

```
if [ condition ]
then
     expressio
n  else
     expression
fi
```

Linux Systems

Shell Scripting - Conditions – if else

```
user@user:PPT-Examples] vi 2_if_then.sh
```

```
  1 #!/bin/bash
  2
  3 NUM1=5
  4 NUM2=3
  5
  6 if [ ${NUM1} -gt ${NUM2} ]
  7 then
  8         echo "NUM1 is greater than NUM2"
  9 else
 10         echo "NUM2 is greater than NUM1"
 11 fi
~
"2_if_then.sh" 11L, 139C                                    0,0-1         All
```

```
user@user:PPT-Examples] chmod +x 2_if_then.sh
user@user:PPT-Examples] ./2_if_then.sh
NUM1 is greater than NUM2
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting - Conditions – if else if

- Multiple if blocks can be strung together to make an elaborate set of conditional responses

Syntax

```
if [ condition_a ]
then
      condition_a is true
elif [ condition_b ]
then
      condition_b is true
else
      both false
fi
```

Linux Systems

Shell Scripting - Conditions – if else if

```
user@user:PPT-Examples] vi 3_if_elif.sh
```

```
  1 #!/bin/bash
  2
  3 NUM1=5
  4 NUM2=3
  5
  6 if [ ${NUM1} -eq ${NUM2} ]
  7  then
  8        echo   "NUM1 is equal to NUM2"
  9  elif [ ${NUM1} -gt ${NUM2} ]
 10  then
 11        echo   "NUM1 is greater than NUM2"
 12 else
 13        echo   "NUM1 is less than NUM2"
 11 fi
~
"3_if_elif.sh" 14L, 200C                                  0,0-1         All
```

```
user@user:PPT-Examples] chmod +x 3_if_elif.sh
user@user:PPT-Examples] ./3_if_elif.sh
NUM1 is less than NUM2
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting - Conditions – case

- The case statement compares the value of the variable ($var in this case) to one or more values
- Once a match is found, the associated commands are executed and the case statement is terminated
- Used to execute statements based on specific values
- Often used in place of an if statement if there are a large number of conditions.
- Each set of statements must be ended by a pair of semicolon
- *) is used for not matched with list of values

Syntax

```
case ${VAR}
     in
     value_1)
          commands;
          ;;
     value_2)
          commands;
          ;;
     *)
          commands;
          ;;
esac
```

Linux Systems

Shell Scripting - Conditions – case

```
user@user:PPT-Examples] vi 4_case_statments.sh
```

```
 1 #!/bin/bash
 2
 3   echo “Enter a number:”
 4read NUM
 5
 6   case ${NUM} in
 7           1)
 8                 echo "You entered One"
 9                 ;;
10           2) echo "You entered Two"
11                 ;;
12           *) echo "Obey my orders please" ;;
13 esac
~
~
"4_case_statements.sh" 13L, 187C                              0,0-1          All
```

Linux Systems

Shell Scripting - Conditions – case

```
user@user:PPT-Examples] chmod +x 4_case_statements.sh
user@user:PPT-Examples] ./4_case_statements.sh   "Enter a
number < 3:"
2
You entered Two
user@user:PPT-Examples] ./4_case_statements.sh
"Enter a number < 3:"
5
Obey my orders please
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting – Numeric Test Operators

Operator	Functions
-eq	Compare if two numbers are equal
-ge	Compare if one number is greater than or equal to num
-le	Compare if one number is less than or equal to a num
-ne	Compare if two numbers are not equal
-lt	Compare if one number is less than another number
-gt	Compare if one number is greater than another number

Linux Systems

Shell Scripting - String Tests

- String comparison, Numeric comparison, File operators and logical operators
- Comparison operations are provided below

Operator	Functions
=	Compare if two strings are equal
!=	Compare if two strings are not equal
-n	Evaluate if string length is greater than zero
-z	Evaluate if string length is equal to zero

Linux Systems

Shell Scripting - String Tests

user@user:PPT-Examples] vi 5_string_test.sh

```
 1 #!/bin/bash
 2
 3 echo "Enter the      first string"
 4 read STR1
 5 echo "Enter the      second string"
 6read STR2
 7
 8 if [ -z${STR1}       ]; then
 9         echo "First string is empty"
10 else
11         echo "First string is not empty"
12 fi
13 if [    -n ${STR2} ]; then
14         echo "Second string is not empty"
15 else
16         echo "Second string is empty"
17 fi
18 if [    ${STR1} = ${STR2} ]; then
19         echo "Both strings are equal"
20 else
21         echo "Both strings are not equal"
22 fi
~
"5_string_test.sh" 22L, 405C                                  0,0-1          All
```

Linux Systems

Shell Scripting - String Tests

```
user@user:PPT-Examples] chmod +x 5_string_test.sh
user@user:PPT-Examples] ./5_string_test.sh  "Enter the
first string"
Hello
"Enter the second string"
World
First string is not empty
Second string is not empty
Both strings are not equal
user@user:PPT-Examples] ./5_string_test.sh
"Enter the first string"
Hello
"Enter the second string"
Hello
First string is not empty
Second string is not empty
Both strings are equal
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting – Logical Operators

Operator	Functions
!	Compare if two strings are equal
-a	Logically AND two logical expression
-o	Logically OR two logical expressions

user@user:PPT-Examples] vi 6_logical_operator.sh

```
 1 #!/bin/bash
 2
 3  echo “Enter the first number A” ; read A
 4  echo “Enter the second number B” ; read B
 5echo “Enter the third number C” ; read C 6
 7 if [ ${A} -gt ${B} -a ${A} -gt ${C} ]; then
 8         echo "A is the greatest of all"
 9 elif [ ${B} -gt ${A} -a ${B} -gt ${C} ]; then
10         echo "B is the greatest of all"
14 elif [ ${C} -gt ${A} -a ${C} -gt ${B} ]; then

15         echo "C is the greatest of all"
16 else
17         echo "Invalid Input"
18 fi
"6_logical_operators.sh" 18L, 426C                    0,0-1          All
```

Linux Systems

Shell Scripting - Logical Operator

```
user@user:PPT-Examples] chmod +x 6_logical_operator.sh
user@user:PPT-Examples] ./6_logical_operator.sh   "Enter the
first number A"
10
"Enter the second number B "
2
"Enter the third number C "
3
A.is the greatest of all
user@user:PPT-Examples] ./6_logical_operator.sh
"Enter the first number A"
3
"Enter the second number B "
10
"Enter the third number C "
2
B.is the greatest of all
user@user:PPT-Examples]
```

File Specific Commands and Operators

Linux Systems

File Specific Cmds & Opers - more

- **more** helps us to view a file content page wise

```
user@user:PPT-Examples] more 2_if_then.sh # Use q to quit
#!/bin/bash

NUM1=
15
NUM2=
6

if [ ${NUM1} -gt ${NUM2} ]
then
        echo    "NUM2 is greater than NUM1"
fi
user@user:PPT-Examples]
```

- Note, if the file size if greater than the window size you would have page view
- You may try the below example

```
user@user:PPT-Examples] sudo more /var/log/syslog
```

Linux Systems

File Specific Cmds & Opers - less

- **less** similar to **more** with many features

```
user@user:PPT-Examples] less 2_if_then.sh # Use q to quit
```

```
#!/bin/bash

NUM1=
15
NUM2=
6

if [ ${NUM1} -gt ${NUM2} ]
then
        echo    "NUM1 is greater than NUM2"
else
        echo    "NUM2 is greater than NUM1"
fi
2_if_then.sh (END)
```

- Note, if the file size if greater than the window size you would have page view
- You may try the below example

```
user@user:PPT-Examples] sudo less /var/log/syslog
```

Linux Systems

File Specific Cmds & Opers - | (pipe)

- A pipe is a form of redirection that is used in Linux operating systems to send the output of one program to another program for further processing.
- A pipe is designated in commands by the vertical bar character

```
user@user:PPT-Examples] ls
10_cmd_line_args.sh      4_string_test.sh                8_while_loop.sh
1_expressions.sh  2_if_the5n_.lsohgical_operators.sh     9_arrays.sh
3_if_elif.sh             6_case_statements.sh
                         7_for_loop.sh
user@user:PPT-Examples] ls | wc # Gives the word count info from ls output   10
                10       166
user@user:PPT-Examples]
```

- It is a very useful operator. Well see some more usage in later slide

Linux Systems

File Specific Cmds & Opers - head

- **head** helps us to output the first part of files

```
user@user:PPT-Examples]    head -5 1_expressions.sh    # Print   first    5 lines
#!/bin/bash

NUM1
=5
NUM2
=3

user@user:PPT-Examples]
```

```
user@user:PPT-Examples]    ls -1v | head -5 # Print      first 5    lines    for ls output
1_expressions.sh
2_if_then.sh  3_if_elif.sh
4_string_test.sh
5_logical_operators.sh
user@user:PPT-Examples]
```

Linux Systems

File Specific Cmds & Opers - tail

- **tail** helps us to output the last part of files

```
user@user:PPT-Examples] tail -5 1_expressions.sh # Print last 5 lines   echo -e
"Addition of two numbers is\t\t: ${ADD}"
echo -e "Substraction of two numbers is\t\t: ${SUB}"   echo -
e "Multiplication of two numbers is\t: ${MUL}"   echo -e
"Division of two numbers is\t\t: ${DIV}"   echo -e "Modulum
of two numbers is\t\t: ${MOD}"   user@user:PPT-Examples]
```

```
user@user:PPT-Examples] ls -1v | tail -5 # Print first 5 lines for ls output   7_for_loop.sh
7_while_loop.sh
6_case_statements.sh
9_arrays.sh
10_cmd_line_args.sh
user@user:PPT-Examples]
```

Linux Systems

File Specific Cmds & Opers - Redirection

- Operators used to redirect data
  - From a file to a command – Input Redirection – Data <
  - To a file from the command – Output Redirection – Data >
    - We may control the data stream which passed to the file
      - 1> – Data from stdout to file
      - 2> – Data from stdin file
    - Data can be appended to an existing file
      - >>
- The functions discussed here basic
- Many advanced functions are available if required

Linux Systems

File Specific Cmds & Opers - Redirection

```
user@user:PPT-Examples] ls
10_cmd_line_args.sh     4_string_test.sh          8_while_loop.sh
1_expressions.sh        5_logical_operators.sh    9_arrays.sh
2_if_then.sh            6_case_statements.sh
3_if_elif.sh            7_for_loop.shh
user@user:PPT-Examples] echo Hello > test.txt
user@user:PPT-Examples] ls
10_cmd_line_args.sh     4_string_test.sh          8_while_loop.sh
1_expressions.sh        5_logical_operators.sh    9_arrays.sh
2_if_then.sh            6_case_statements.sh      test.txt
3_if_elif.sh            7_for_loop.sh
user@user:PPT-Examples] cat text.txt
Hello
user@user:PPT-Examples] echo World > test.txt # Overwrites the files
user@user:PPT-Examples] cat text.txt
World
user@user:PPT-Examples] echo Hello >> test.txt
user@user:PPT-Examples] cat text.txt
Worl
d
Hello
```

Linux Systems

File Specific Cmds & Opers - Redirection

```
user@user:PPT-Examples] ls
10_cmd_line_args.sh       4_string_test.sh              8_while_loop.sh
1_expressions.sh      5_logical_operators.sh        9_arrays.sh  test.txt
2_if_then.sh              6_case_statements.sh
3_if_elif.sh              7_for_loop.sh
user@user:PPT-Examples] ls 9_arrays.sh 1234 > test.txt  ls:
cannot access '1234': No such file or directory
user@user:PPT-Examples] cat text.txt
9_arrays.sh
user@user:PPT-Examples] ls 9_arrays.sh 1234 2> test.txt
9_arrays.sh
user@user:PPT-Examples] cat text.txt
ls: cannot access '1234': No such file or directory
user@user:PPT-Examples] echo 1 + 2 > text.txt
user@user:PPT-Examples] bc < test.txt # The file contents is passed as input
3
user@user:PPT-Examples] rm text.txt # Not needed anymore
user@user:PPT-Examples]
```

Linux Systems

File Specific Cmds & Opers - du

- **du** helps us to estimate file space usage

```
user@user:PPT-Examples] du
44      .
user@user:PPT-Examples] du -h # Human readable
44K     .
user@user:PPT-Examples] du -sb * # Size in bytes
301 10_cmd_line_args.sh
416 1_expressions.sh
139 2_if_then.sh
200 3_if_elif.sh
405 4_string_test.sh
427 5_logical_operators.sh
187 6_case_statements.sh
131 7_for_loop.sh
103 8_while_loop.sh
436 9_arrays.sh
user@user:PPT-Examples]
```

Linux Systems

File Specific Cmds & Opers - df

- **df** helps us to fine system disk space usage

```
user@user:PPT-Examples] df
tmpfs                803976       952    794456   2% /run
/dev/sda1          76765204    955756  63285052  14% /
tmpfs                401986         4    401980   1% /dev/shm
tmpfs                401986         0    401986   0% /sys/fs/cgroup
/dev/sda6          519987072   7165932  42184752  15% /home
tmpfs                803976        60    803916   1% /run/user/1001
user@user:PPT-Examples] df -h # Human readable
udev                 3.9G     0  3.9G   0% /dev
tmpfs                786M   9.3M  776M   2% /run
/dev/sda1             74G   9.2G   61G  14% /
tmpfs                3.9G   8.3M  3.9G   1% /dev/shm
tmpfs                5.0M   4.0K  5.0M   1% /run/lock
/dev/sda6            496G    69G  403G  15% /home
tmpfs                786M    60K  786M   1% /run/user/1001
```

Linux Systems

File Specific Cmds & Opers - df

- **df** helps us to find system disk space usage

```
user@user:PPT-Examples] df # Note the output depends on the installation
tmpfs                  803976     9520     794456   2% /run
/dev/sda1            76765204 63285052  9557568  14% /
tmpfs                4019868     8484    4011384   1% /dev/shm
tmpfs                   5120        4       5116   1% /run/lock
tmpfs                4019868        0    4019868   0% /sys/fs/cgroup
/dev/sda6          519987072 71655932  421847532  15% /home
tmpfs                 803976       60     803916   1% /run/user/1001
user@user:PPT-Examples] df -h # Human readable

/dev/sda1              74G  9.2G   61G  14% /
tmpfs                 3.9G  8.3M  3.9G   1% /dev/shm
tmpfs                 5.0M  4.0K  5.0M   1% /run/lock
/dev/sda6             496G   69G  403G  15% /home
tmpfs                 786M   60K  786M   1% /run/user/1001
```

Linux Systems

File Specific Cmds & Opers - stat

- **stat** helps us to display file or file system status

```
user@user:PPT-Examples] stat 1_expression.sh
  File: '1_expressions.sh'
  Size: 416             Blocks: 8          IO Block: 4096   regular file
Device: 806h/2054d Inode:                  Links: 1
A23cc6e0s0s9:7(30777/-rwxrwxrwx)Uid: ( 1001/ adil_sk)   Gid: ( 1001/ adil_sk)
Access: 2018-12-03 16:24:18.489886098 +0530
Modify: 2018-12-03 14:24:00.896270594 +0530
Change: 2018-12-03 14:29:10.811075929 +0530
 Birth: -
user@user:PPT-Examples]
```

Linux Systems

File Specific Cmds & Opers - ln

- **ln** helps us to make links between file
- Two types of links are possible
  - Hard link → Default option
    - The link file has the same inode number
    - Doesn't matter if the original file get deleted
    - It a default link created
  - Soft link → Default option files and directories
    - The link file has the different inode number
    - If the original file is deleted the link get broken
    - Created with **-s** option

Linux Systems

File Specific Cmds & Opers - ln - Hard link

```
user@user:PPT-Examples] echo Hello > test.txt
user@user:PPT-Examples] cat test.txt
Hello
user@user:PPT-Examples] ln test.txt hard_link.txt # Default is hardlink
user@user:PPT-Examples] cat hard_link.txt
Hello
user@user:PPT-Examples] echo World >> hard_link.txt
user@user:PPT-Examples] cat text.txt
Hello
Worl
d
user@user:PPT-Examples] stat text.txt | head -3
  File: 'text.txt'
  Size: 6               Blocks: 8          IO Block: 4096   regular file
Device: 806h/2054d      Inode: 23600979    Links: 2
user@user:PPT-Examples] stat hard_link.txt | head -3
  File: 'hard_link.txt'
  Size: 6               Blocks: 8          IO Block: 4096   regular file
Device: 806h/2054d      Inode:             Links: 2
user@user:PPT-Examples] rm text.txt # Doesn't effect the hard link!!
user@user:PPT-Examples] cat hard_link.txt
Hello
Worl
d
```

Linux Systems

File Specific Cmds & Opers - ln - Soft link

```
user@user:PPT-Examples] echo Hello > test.txt
user@user:PPT-Examples] ln -s test.txt soft_link.txt
user@user:PPT-Examples] ls -l soft_link.txt
lrwxrwxrwx 1 user user 8 Dec        4 15:01 soft_link.txt -> test.txt
user@user:PPT-Examples] cat soft_link.txt
Hello
user@user:PPT-Examples] echo World >> soft_link.txt
user@user:PPT-Examples] cat text.txt
Hello
Worl
d
user@user:PPT-Examples] stat text.txt | head -3
  File: 'text.txt'
  Size: 6               Blocks: 8          IO Block: 4096   regular file
Device: 806h/2054d      Inode:             Links: 1
user@user:PPT-Examples] stat soft_link.txt | head -3
  File: 'soft_link.txt'
  Size: 8               Blocks: 8          IO Block: 4096   regular file
Device: 806h/2054d      Inode:             Links: 1
user@user:PPT-Examples]rm text.txt # This effects the soft link!!
user@user:PPT-Examples] cat soft_link.txt
cat: soft_link.txt: No such file or directory
user@user:PPT-Examples] ls -l soft_link.txt # Broken link!!  lrwxrwxrwx 1
user user 8 Dec                    4 15:01 soft_link.txt -> test.txt
user@user:PPT-Examples]
```

Shell Scripting – Part 2

Linux Systems

Shell Scripting - Loops – for

- The structure is a looping structure. Used to execute a set of commands while the provided list is empty.
- The loop terminates as soon as the all the elements in the list is evaluated.
- Can be used in multiple methods, an example for fixed iteration is show below

Syntax

```
for i in list
do
    Code
Block  done
```

Linux Systems

Shell Scripting - Loops – for

```
user@user:PPT-Examples] vi 7_for_loop.sh
```

```
  1 #!/bin/bash
  2
  3 for i in 1 2 3 4 5
  4 do
  5             echo    "Loop counter is ${i}"
  6 done
~
"7_for_loop.sh" 6L, 70C                                       0,0-1          All
```

```
user@user:PPT-Examples]   chmod   +x   7_for_loop.sh
user@user:PPT-Examples] ./7_for_loop.sh
Loop counter is 1
Loop counter is 2  Loop
counter is 3  Loop
counter is 4  Loop
counter is 5
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting - Loops – while

- The structure is a looping structure. Used to execute a set of commands while a specified condition is true

- The loop terminates as soon as the condition becomes false. If condition never becomes false, loop will never exit

- Any valid conditional expression will work in the while loop.

Syntax

```
while [ condition ]
do
     Code
Block  done
```

Linux Systems

Shell Scripting - Loops – while

```
user@user:PPT-Examples] vi 8_while_loop.sh
```

```
  1 #!/bin/bash
  2
  3
  LOOP=1
  4
  5 while [ ${LOOP} -le 5]
  7 do          echo    "Looping : ${LOOP}"
  8             LOOP=$((${LOOP} + 1))
  9 done
~
"8_while_loop.sh" 9L, 103C                                    0,0-1          All
```

```
user@user:PPT-Examples] chmod +x 8_while_loop.sh
user@user:PPT-Examples] ./8_while_loop.sh   Looping :
1
Looping : 2
Looping : 3
Looping : 4
Looping : 5
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting -Arrays

- An array is a variable containing multiple values may be of same type or of different type
- There is no maximum limit to the size of an array
- Array index starts with zero

Linux Systems

Shell Scripting -Arrays

```
user@user:PPT-Examples] vi 9_arrays.sh
```

```
  1  #!/bin/bash
  2  LINUX_DISTROS=('Debian' 'Redhat' 'Ubuntu' 'Suse' 'Fedora');
  3  echo "Number of elements in the array: ${#LINUX_DISTROS[@]}"
  4  echo -e "Printing elements of array in one shot\t\t: ${LINUX_DISTROS[@]}"
  5  echo -e "Printing elements of array in one shot\t\t: ${LINUX_DISTROS[*]}"
  6  echo -en "Printing elements of array in using a loop\t:"
  7  for i in ${LINUX_DISTROS[@]}
  8  do
  9       echo -n ${i}" "
  10 done; echo
~
"9_arrays.sh" 10L, 436C                                    0,0-1          All
```

```
user@user:PPT-Examples] chmod +x 9_arrays.sh
user@user:PPT-Examples] ./9_arrays.sh  Number
of elements in the array: 5
Printing elements of array in one shot : Debian Redhat Ubuntu Suse Fedora
Printing elements of array in one shot : Debian Redhat Ubuntu Suse Fedora
Printing elements of array in using a loop : Debian Redhat Ubuntu Suse Fedora
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting – Command LineArguments

- Shell script can accept command-line arguments & options just like other Linux commands
- Within your shell script, you can refer to these arguments as $1,$2,$3,.. & so on.
- Then the command line arguments are executed like
- Read all command line arguments and print them

Linux Systems

Shell Scripting - Command LineArguments

```
user@user:PPT-Examples] vi 10_cmd_line_args.sh
```

```
  1  #!/bin/bash
  2
  3  if [ $# != 2 ]
  4  then
  5      echo "Usage: Pass 2 arguments"
  6      exit 0
  7fi  8
  9  echo "The arguments of the script you passed are:"
  10 echo "Total number of arguments you passed are : $#"

  11 echo "The name of the script is                  : $0"
  12 echo "The first argument is                      : $1"
  13 echo "The second argument is                     : $2"
~
"10_cmd_line_args.sh" 13L, 301C                      0,0-1          All
```

Linux Systems

Shell Scripting - Command LineArguments

```
user@user:PPT-Examples] chmod +x 10_cmd_line_args.sh
user@user:PPT-Examples] ./10_cmd_line_args.sh Hello 1234
The arguments of the script you passed are:
Total number of arguments you passed are          : 2
The name of the script is                         : ./10_cmd_line_args.sh
The first argument is                             : Hello
The second argument is                            : 1234
user@user:PPT-Examples]
```

Linux Systems

Shell Scripting - Functions

- Writing functions can greatly simplify a program
- Improves modularity, readability and maintainability
- However speed will get slowed down
- Arguments are accessed as $1, $2, $3…

Syntax

```
function name()
{
    <command>
    <statments>
    <expression>
}
```

Linux Systems

Shell Scripting - Functions

```
user@user:PPT-Examples] vi 11_functions.sh
```

```
  1  !/bin/bash
  2
  3  function sum()
  4  {
  5      x=`expr $1 + $2`
  6      echo $x
  7  }
  8
  9  y=`sum 5 3`
 10 echo "The sum is 5 and 3 is $y"
 11 echo "The sum is 6 and 2 is `sum 6 2`"
~
"11_functions.sh" 11L, 146C                                    0,0-1         A
```

```
user@user:PPT-Examples] chmod +x 11_functions.sh
user@user:PPT-Examples] ./11_functions.sh
The sum is 5 and 3 is8
The sum is 6 and 2 is8
user@user:PPT-Examples]
```

User Specific Commands

Linux Systems

User Specific Commands – useradd and su

- All Accesses into a Linux System are through a User
- Super user (root) will have higher privileges
- **useradd** helps us to create a new user or update default new user information

```
user@user:~] useradd pingu
useradd: Permission denied.
useradd: cannot lock /etc/passwd; try again later.
user@user:~] sudo useradd pingu
[sudo] password for user:
user@user:~]
```

- **su** to change user ID or become superuser

```
user@user:~] su - pingu # Note, we did not set the password yet
Password:
```

- We need to set the password!. Refer the next slide

Linux Systems

User Specific Commands -passwd

- So, lets set the password for the user using **passwd** which is used to change user password
- Terminate the prompt by **CTRL D** and follow the below command

```
user@user:~] passwd pingu
passwd: You may not view or modify password information for pingu.
user@user:~] sudo passwd # Password won't be visible, its shadow password
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
user@user:~] su -pingu
Password:
No directory, logging in with HOME=/
$ # We have created the user without any option, hence the above message,
You may see man useradd for more information
$ exit # Exit to the previous user
user@user:~]
```

- One simple method to solve the above issue is to re create the user again

Linux Systems

User Specific Commands – userdel

- **userdel** to delete a user account and related files

```
user@user:~] userdel pingu
userdel: Permission denied.
userdel: cannot lock /etc/passwd; try again later.
user@user:~] sudo userdel pingu
[sudo] password for user:
user@user:~]
```

Linux Systems

User Specific Commands – w, who, whoami

- **w** shows who is logged on and what they are doing

```
user@user:~] w
 23:06:47 up  4:47,  8 users,  load average: 0.43, 0.48, 0.45
USER     TTY      FROM      LOGIN@   IDLE   JCPU   PCPU WHAT
user     tty1     :0        18:20    4:46m  7:18   0.11s /bin/sh
user     pts/0              18:20    1.00s  0.31s  0.00s w
user     pts/1    :0        18:20   16:52   3.39s  3.39s /bin/bash
user     pts/2              18:20    4:45m  0.00s 13.43s kded5 [kdeinit5]
user@user:~]
```

- **who** shows who is logged on

```
user@user:~] who
adil     tty1         2018-12-06 18:20 (:0)
adil     pts/0        2018-12-06 18:20 (:0)
adil     pts/1        2018-12-06 18:20 (:0)
adil     pts/2        2018-12-06 18:20 (:0)
user@user:~]
```

- **whoami** prints effective userid

```
user@user:~] whoami
user
user@user:~]
```

Linux Systems

User Specific Commands – scp

- **scp** secure copy (remote file copy program)

```
user@user:~] echo Hello > test.txt
user@user:~] scp test.txt tingu@192.168.1.100:
The authenticity of host '192.168.1.103 (192.168.1.100)' can't be established.
ECDSA key fingerprint is SHA256:1IaykOSlwAeeEBk2XgmoxS0Ys51ORx9wQX4CiwkWI9o.
Are you sure you want to continue connecting (yes/no)?yes
```

```
Warning: Permanently added '192.168.1.100' (ECDSA) to the list of known hosts.
tingu@192.168.1.100's password:
```

```
test.txt                                      100%    6    10.2KB/s   00:00
user@user:~]
```

- The authenticity message come whenever there is new user is found
- You will have to add **-r** in case of directories
- You may provide the destiny path after the **:** so that the file gets copied in the provided path

Linux Systems

User Specific Commands – ssh

- **ssh** OpenSSH SSH client (remote login program)

```
user@user:~] ssh tingu@192.168.1.100
tingu@192.168.1.100's password:
Welcome to Ubuntu 18.10 (GNU/Linux 4.18.0-12-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

0 packages can be updated.
0 updates are security updates.

Last login: Tue Aug  7 12:42:46 2018
tingu@my-lappi:~$
```

```
tingu@my-lappi:~$ logout

Connection to 192.168.1.100 closed.
user@user:~]
```

- Well both scp and ssh requires some network information like and all, so lets see some basic network related commands

Linux Systems

Network Related Commands – ifconfig

- **ifconfig** to configure a network interface

```
user@user:~] ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 4673  bytes 412341 (412.3 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4673  bytes 412341 (412.3 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp2s0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.103  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 80fe::cd3d:73be:1ac4:32c5  prefixlen 64  scopeid 0x20<link>
        ether A4:82:a2:d5:81:68  txqueuelen 1000  (Ethernet)
        RX packets 31272  bytes 33166704 (33.1 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 18957  bytes 3134847 (3.1 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
user@user:~]
```

Linux Systems

Network Related Commands – ping

- **ping** send ICMP ECHO_REQUEST to network hosts

```
user@user:~] ping 192.168.1.100
PING 192.168.1.100 (192.168.1.100) 56(84) bytes of data.
64 bytes from 192.168.1.100: icmp_seq=1 ttl=64 time=0.049 ms
64 bytes from 192.168.1.100: icmp_seq=2 ttl=64 time=0.052 ms
64 bytes from 192.168.1.100: icmp_seq=3 ttl=64 time=0.051 ms
64 bytes from 192.168.1.100: icmp_seq=4 ttl=64 time=0.052 ms
^C
--- 192.168.1.100 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 76ms
rtt min/avg/max/mdev = 0.045/0.048/0.051/0.002 ms
user@user:~]
```

File Specific Commands

Linux Systems

File Specific Commands – sort

- **sort** lines of text files

```
user@user:~] echo -e "Apple\nOrange\nChicku\nPineapple\nMango" > test.txt
user@user:~] cat test.txt
Apple
Orange
Chicku
Pineapple
Mango
user@user:~] sort test.txt
Apple
Chicku
Mango
Orange
Pineapple
user@user:~] sort -r test.txt
Pineapple
Orange
Mango
Chicku
Apple
user@user:~]
```

- You may even use pipes

Linux Systems

File Specific Commands - uniq

- **uniq** used to report or omit repeated lines

```
user@user:~] echo -e “Apple\nApple\nChicku\n\nApple\nPlum\nMango” > test.txt
user@user:~] cat test.txt # Note the repeated words
Apple
Apple
Chicku
Apple
Plum
Mango
user@user:~] uniq test.txt # Repeated adjacent words are removed
Apple
Chicku
Apple
Plum
Mango
user@user:~] sort test.txt | uniq # One of the method to have unique list
Apple
Chicku
Mango
Plum
user@user:~]
```

Linux Systems

File Specific Commands – cmp

- **cmp** to compare two files byte by byte

```
user@user:~] echo -e “Apple\nBanana” > test1.txt
user@user:~] echo -e “apple\nBanana” > test2.txt
user@user:~] cmp test1.txt test2.txt
test1.txt test2.txt differ: byte 1, line 1
user@user:~] echo -e “Apple\nbanana” > test2.txt
user@user:~] cmp test1.txt test2.txt
test1.txt test2.txt differ: byte 7, line 2
user@user:~] echo -e “apple\nBanana” > test1.txt
user@user:~] cmp test1.txt test2.txt # Note there are 2 changes, it reports
1st change
test1.txt test2.txt differ: byte 1, line 1
user@user:~]
```

Linux Systems

File Specific Commands – diff

- **diff** to compare files line by line

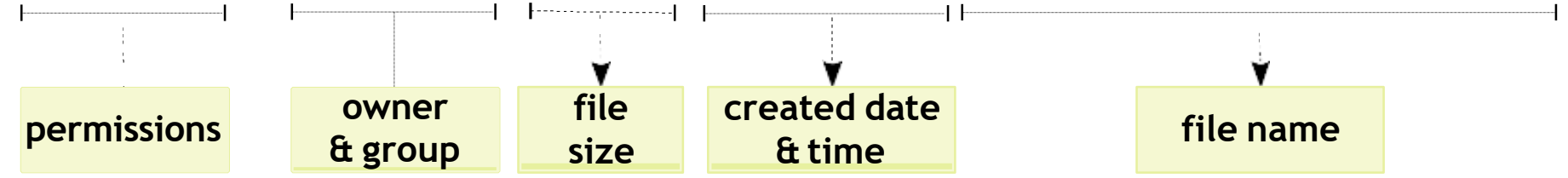
```
user@user:~] echo -e “Apple\nBanana” > test1.txt
user@user:~] echo -e “apple\nBanana” > test2.txt
user@user:~] diff test1.txt test2.txt
1,2c1,2
< apple
< Banana
---
> Apple
> banana
user@user:~]
```

Linux Systems

Types of Files

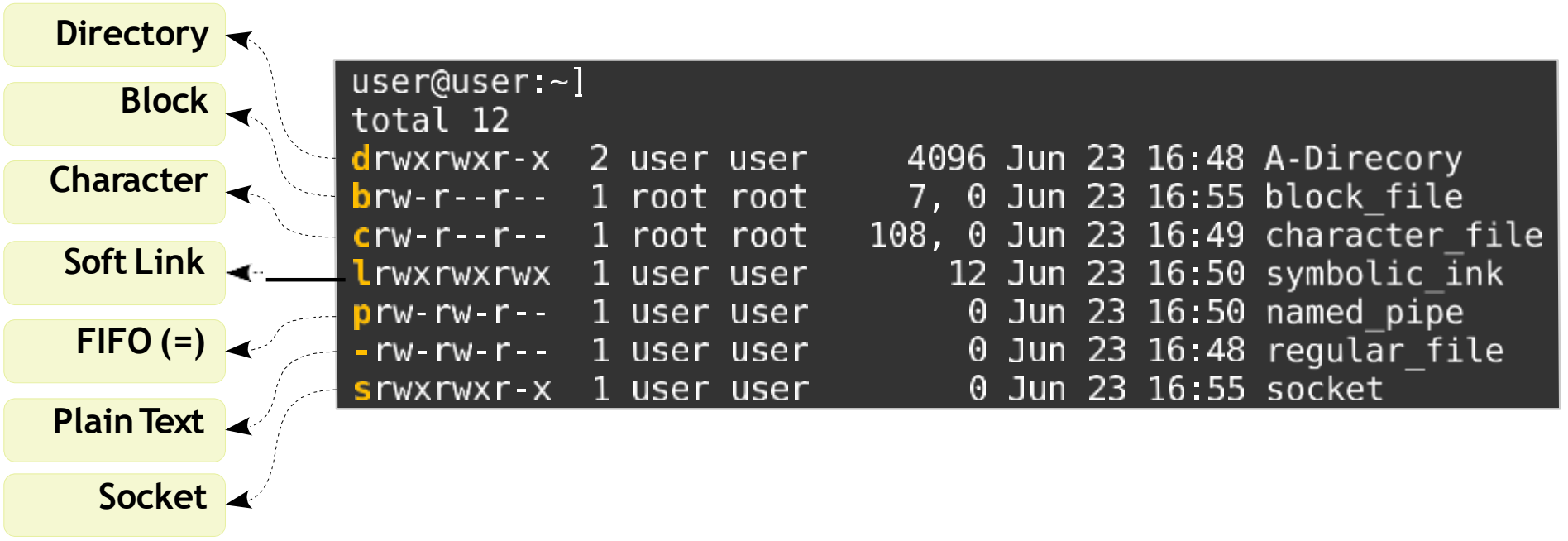
- In Linux system there are different types of files
- Every file belongs to different group and has different permissions
- The below example shows the different types of files available in Linux

```
user@user:~]
total 12
drwxrwxr-x  2 user user     4096 Jun 23 16:48 A-Direcory
brw-r--r--  1 root root     7, 0 Jun 23 16:55 block_file
crw-r--r--  1 root root   108, 0 Jun 23 16:49 character_file
lrwxrwxrwx  1 user user       12 Jun 23 16:50 symbolic_ink -> regular_file
prw-rw-r--  1 user user        0 Jun 23 16:50 named_pipe
-rw-rw-r--  1 user user        0 Jun 23 16:48 regular_file
srwxrwxr-x  1 user user        0 Jun 23 16:55 socket
```



Linux Systems

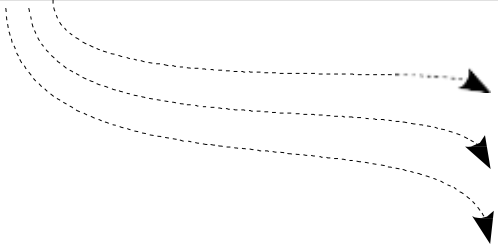
Types of Files



Linux Systems

Types of Files

```
user@user:~]
total 12
drwxrwxr-x  2 user user     4096 Jun 23 16:48 A-Direcory
brw-r--r--  1 root root     7, 0 Jun 23 16:55 block_file
crw-r--r--  1 root root   108, 0 Jun 23 16:49 character_file
lrwxrwxrwx  1 user user       12 Jun 23 16:50 symbolic_ink
prw-rw-r--  1 user user        0 Jun 23 16:50 named_pipe
-rw-rw-r--  1 user user        0 Jun 23 16:48 regular_file
srwxrwxr-x  1 user user        0 Jun 23 16:55 socket
```



x → Execute 001 - 1

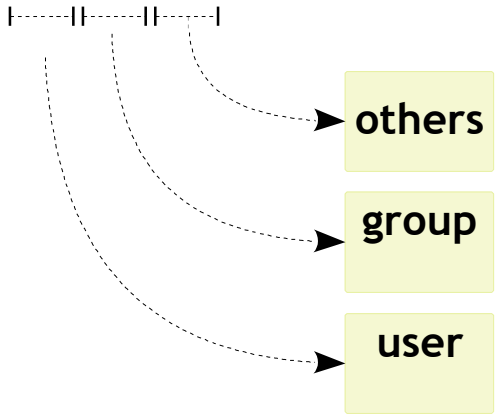
w → Write 010 - 2

r → Read 100 - 4

Linux Systems

Types of Files

```
user@user:~]
total 12
drwxrwxr-x  2 user user     4096 Jun 23 16:48 A-Direcory
brw-r--r--  1 root root     7, 0 Jun 23 16:55 block_file
crw-r--r--  1 root root   108, 0 Jun 23 16:49 character_file
lrwxrwxrwx  1 user user       12 Jun 23 16:50 symbolic_ink
prw-rw-r--  1 user user        0 Jun 23 16:50 named_pipe
-rw-rw-r--  1 user user        0 Jun 23 16:48 regular_file
srwxrwxr-x  1 user user        0 Jun 23 16:55 socket
```



Linux Systems

File Specific Commands - chmod

- **chmod** used to change file mode bits

```
user@user:~]  touch test.txt  ls
user@user:~]  -l test.txt
  -rw-rw-r-- 1 user user 0 Dec 10     18:40   test.txt
user@user:~] chmod 777 test.txt
user@user:~] ls -ltest.txt
 -rwxrwxrwx user user 0 Dec10     18:40   test.txt
              1
user@user:~] chmod 707 test.txt
user@user:~] ls -ltest.txt
  -rwx---rwx 1 user user 0 Dec 10     18:40   test.txt
user@user:~] chmod -xtest.txt
user@user:~]  ls -ltest.txt
  -rw-rw-rw- 1 user user 0 Dec 10     18:40   test.txt
user@user:~] chmod o+x test.txt
user@user:~] ls -ltest.txt
 -rw-rwxrw- 1 user user 0 Dec 10     18:40   test.txt
user@user:~] chmod  a-r test.txt
user@user:~] ls -ltest.txt
 --w--wx-w- 1 user user 0 Dec 10     18:40   test.txt
user@user:~]
```

Linux Systems

File Specific Commands - chown

- **chown** used to file owner and group

```
user@user:~] ls -l test.txt
-rw-rw-r-- 1 user user 0 Dec 10 18:40 test.txt
user@user:~] sudo chown user.nobody test.txt
user@user:~] ls -l test.txt
-rw-rw-r-- 1 user nobody 0 Dec 10 18:40 test.txt
user@user:~] mkdir TBD # To Be Deleted
user@user:~] touch TBD/{1..4}.txt # Just create 4 files
user@user:~] ls -l TBD
total 0
-rw-rw-r-- 1 user user 0 Dec 10 19:01 1.txt
-rw-rw-r-- 1 user user 0 Dec 10 19:01 2.txt
-rw-rw-r-- 1 user user 0 Dec 10 19:01 3.txt
-rw-rw-r-- 1 user user 0 Dec 10 19:01 4.txt
user@user:~] sudo chown user.nobody -R TBD # Recursive, since directory
user@user:~] ls -l TBD
total 0
-rw-rw-r-- 1 user nobody 0 Dec 10 19:01 1.txt
-rw-rw-r-- 1 user nobody 0 Dec 10 19:01 2.txt
-rw-rw-r-- 1 user nobody 0 Dec 10 19:01 3.txt
-rw-rw-r-- 1 user nobody 0 Dec 10 19:01 4.txt
user@user:~] rm -fr TBD/ test.txt # Just remove the stray contents
user@user:~]
```

Linux Systems

Regular Expression

- Regular expressions = search (and replace / modify / remove) pattern
- In theoretical computer science regular expressions are called as regex or regexp
- It is a sequence of characters that forms a search pattern using some special characters
- Popular applications in Linux (Vi editor, Grep, Sed, Lex & Yacc etc..) extensively use regular expressions
- Extensively used in compiler design and implementation
- Our idea is to understand them from Linux commands

Linux Systems

Regular Expression

- Each character in a regular expression is either understood to be a meta-character with its special meaning
- Or a regular character with its literal meaning
- Together they form a pattern. Some popular & most frequently used examples are provided below

Meta-character	Meaning
?	Zero or one occurrence
*	Zero or more occurrence
+	One or more occurrence

Linux Systems

Search Commands - find

- **find** to search for files in a directory hierarchy

```
user@user:~] mkdir -p TBD/Dir1
user@user:~] touch TBD/Dir1/{1..4}.txt
user@user:~] mkdir TBD/Dir2
user@user:~] echo hello > TBD/Dir2/non_empty_file.txt
user@user:~] echo ls > TBD/Dir2/my_ls.sh  user@user:~]
chmod +x TBD/Dir2/my_ls.sh
user@user:~] cd TBD
user@user:TBD] find . -print # Prints the contents of currentdirectory
.
./Dir2
./Dir2/non_empty_file.txt
./Dir2/my_ls.sh
./Dir1
./Dir1/1.txt
./Dir1/4.txt
./Dir1/2.txt
./Dir1/3.txt
user@user:TBD] find . -name 1.txt
./Dir1/1.txt
user@user:TBD] find . -name *.sh
./Dir2/my_ls.sh
user@user:TBD]
```

Linux Systems

Search Commands - find

```
user@user:TBD] find . -empty
./Dir1/1.txt
./Dir1/4.txt
./Dir1/2.txt
./Dir1/3.txt
user@user:TBD] find . ! -empty
.
./Dir2
./Dir2/non_empty_file.txt
./Dir2/my_ls.sh
./Dir1
user@user:TBD] find . -type f -executable
./Dir2/my_ls.sh
user@user:TBD] find ../ -type d -name Dir1
../TBD1/Dir1
user@user:TBD]
```

Linux Systems

Search Commands – grep

- **grep** to print lines matching a pattern
- Get Regular Expression And Print (GREP)
- Is a pattern matching tool used to search the name input file

```
user@user:TBD]  echo -e "Apple\nGrapes\nBanana" > Dir1/1.txt
user@user:TBD]  echo -e "Raw Banana\nCarrot\nTomato" > Dir1/2.txt
user@user:TBD]  echo -e "Bangles\nCard\nToothpick" > Dir1/3.txt
user@user:TBD] grep Apple Dir1/1.txt
Apple
user@user:TBD] grep -r Apple . # Search Apple at the given path.
Dir1/1.txt:Apple
user@user:TBD] grep -r Ban # Default path is current diretory
Dir1/1.txt:Banana
Dir1/2.txt:Raw Banana#
Dir1/3.txt:Bangles
user@user:TBD] grep -r Banana
Dir1/1.txt:Banana
Dir1/2.txt:Raw Banana
user@user:TBD] grep -rx Banana # Match only the given pattern
Dir1/1.txt:Banana
user@user:TBD]
```

Linux Systems

Search Commands - grep

```
user@user:TBD] grep -rv Ban # Search everything except Ban
Dir2/non_empty_file.txt:hello
Dir2/my_ls.sh:ls
Dir1/1.txt:Apple
Dir1/1.txt:Grapes
Dir1/2.txt:Carrot
Dir1/2.txt:Tomato
Dir1/3.txt:Bangles
Dir1/3.txt:Card
Dir1/3.txt:Toothpick
user@user:TBD] grep -r Car
Dir1/3.txt:Card
user@user:TBD] grep -ri Car # Ignore case
Dir1/2.txt:carrot
Dir1/3.txt:Card
user@user:TBD] grep -rin Ban # Show line numbers
Dir1/1.txt:3:Banana
Dir1/2.txt:1:Raw Banana
Dir1/3.txt:1:Bangles
user@user:TBD]
```

Linux Systems

Substitute Command - sed

- **sed** is a stream editor for filtering and transforming
- text

  It can be a file, or input from a pipe

```
user@user:TBD] sed `s/Apple/apple/' Dir1/1.txt # Doesn't change in file!
apple
Grape
s
Banan
a
user@user:TBD] cat Dir1/1.txt
Apple
Grapes
Banana
user@user:TBD] sed -i `s/Apple/apple/' Dir1/1.txt # Saves into the file
user@user:TBD] cat Dir1/1.txt
apple
Grapes
Banana
user@user:TBD] sed '3s/a/A/' Dir1/1.txt
apple
Grapes
BAnana
user@user:TBD]
```

Linux Systems

Substitute Command - sed

```
user@user:TBD] sed '3s/a/A/g' Dir1/1.txt # Change globally
apple
Grapes
BAnAn
A
user@user:TBD] echo -e “Mango\nPineapple” >> Dir1/1.txt
user@user:TBD] cat Dir1/1.txt
Apple
Grapes
Banana
Mango
Pineapple
user@user:TBD] sed -n ‘2,4p’ Dir1/1.txt
Grape
s
Banan
a
Mang
o
                sed ‘2,4d’
Apple
Pineappl
e
```

Linux Systems

Substitute Command - cut

- **cut** is used to remove sections from each line of files

```
user@user:TBD] cat > database.txt # Use CTRL-D to end
Tingu, 9783422342, tingu@gmail.com
Pingu, 9744527342, pingu@gmail.com
Zingu, 9993234455, zingu@gmail.com

user@user:TBD] cat database.txt
Tingu, 9783422342, tingu@gmail.com
Pingu, 9744527342, pingu@gmail.com
Zingu, 9993234455, zingu@gmail.com
user@user:TBD] cut -d"," -f1 database.txt # Cut 1st field
Ting
u
Ping
u                                                    field
Zing
u
user@user:TBD] cut -d"," -f2,3 database.txt # Cut 2nd and 3rd
 9783422342, tingu@gmail.com
 9744527342, pingu@gmail.com
 9993234455, zingu@gmail.com
user@user:TBD] cut -c5 database.txt # Cut 5th character
u
u
u
```

Linux Systems

Substitute Command - cut

```
user@user:TBD] cut -c5 --complement database.txt # Get all except 5th char
Ting, 9783422342, tingu@gmail.com
Ping, 9744527342, pingu@gmail.com
Zing, 9993234455, zingu@gmail.com
user@user:TBD] cut -c8-11 database.txt # Get a range from 8th to 11th chars
9783
9744
9993
user@user:TBD]
```

- Many more interesting combinations are possible
- You use the piped outputs to cut

```
user@user:TBD] dmesg | tail -5 | cut -c1-15 # Get only the kernel time stamp
[ 1378.145670]
[ 1378.145685]
[ 1378.145695]
[ 1378.145935]
[ 1378.145952]
user@user:TBD]
```

Linux Systems

Substitute Command - split

- **split** is used to split a file into pieces

```
user@user:TBD] mkdir TBD # Just to collect all the splited files
user@user:TBD] cat database.txt
Tingu, 9783422342, tingu@gmail.com
Pingu, 9744527342, pingu@gmail.com
Zingu, 9993234455, zingu@gmail.com
user@user:TBD] split database.txt TBD/ # Split into files
user@user:TBD] ls TBD
aa
user@user:TBD] split -1 database.txt TBD/ # Split every line into a file
user@user:TBD] ls TBD
aa ab ac
user@user:TBD] cat TBD/aa
Tingu, 9783422342, tingu@gmail.com
user@user:TBD] cat TBD/ac
Zingu, 9993234455, zingu@gmail.com
user@user:TBD] split -b5 database.txt T1BD # Split every 5 bytes into a file
user@user:TBD] ls TBD
aa  ab  ac  ad  ae  af  ag  ah  ai  aj  ak  al  am an  ao  ap  aq  ar
    as at  au
user@user:TBD]   cat TBD/aa # Has 5 bytes from 1st line
Tinguuser@user:TBD] cat TBD/ab # Has next 5 bytes from 1st line and it goes on
, 978user@user:TBD]
```

Linux Systems

Substitute Command - tr

- **tr** translates or deletes characters

```
user@user:TBD] echo hello how are you | tr -d h # Delete all ‘h’  ello ow
are you
user@user:TBD] echo hello how are you | tr [:space:] \\t
Hello	how	are	you
user@user:TBD] echo "my age is 99" | tr -d [:digit:]
my age is
user@user:TBD] tr abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
Hello
HELLO
user@user:TBD] tr ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz
Hello
hello
user@user:TBD] echo "Heelllooo" | tr -s elo
Helo
user@user:TBD]
```

Linux Systems

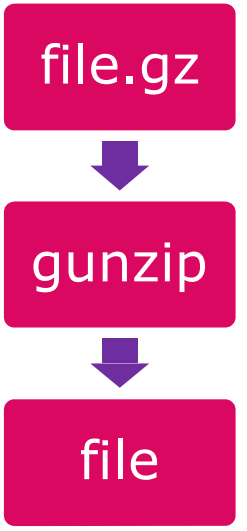
File Compression

- Compression is needed to conserve the disk space
- When there is a need to send large files as an attachment via the email, it is good practice to compress first
- Compression & Decompression utilities - gzip & gunzip(.gz)
- The degree of compression depends on
  - The type of the file
  - Its size
  - Compression program used
- Example
  - Html files compress more
  - GIF & JPEG image files compress very less, as they are already in compressed form

Linux Systems

File Compression - Flow

Compression flow



De-compression flow

- Recursive compression and de-compression (-r option), will come handy
- gzip -r <directory> : To compress files in whole directory
- gunzip -r <directory> : To de-compress files in whole directory

Linux Systems

File Compression – gzip and gunzip

```
user@user:TBD] ls Dir1
Dir2 user@user:TBD] ls
Dir1
1.txt     2.txt     3.txt     4.txt
user@user:TBD] gzip Dir1/1.txt # Compress the file
user@user:TBD] ls Dir1
1.txt.gz     2.txt     3.txt     4.txt
user@user:TBD] gunzip Dir1/1.txt.gz # Decompress the file 1.txt
          2.txt     3.txt     4.txt
user@user:TBD] gzip -r Dir1
1.txt.gz     2.txt.gz     3.txt.gz     4.txt.gz user@user:TBD]
gunzip -r Dir1/ # Decompress the file 1.txt     2.txt       3.txt
              4.txt
user@user:TBD]
```

Linux Systems

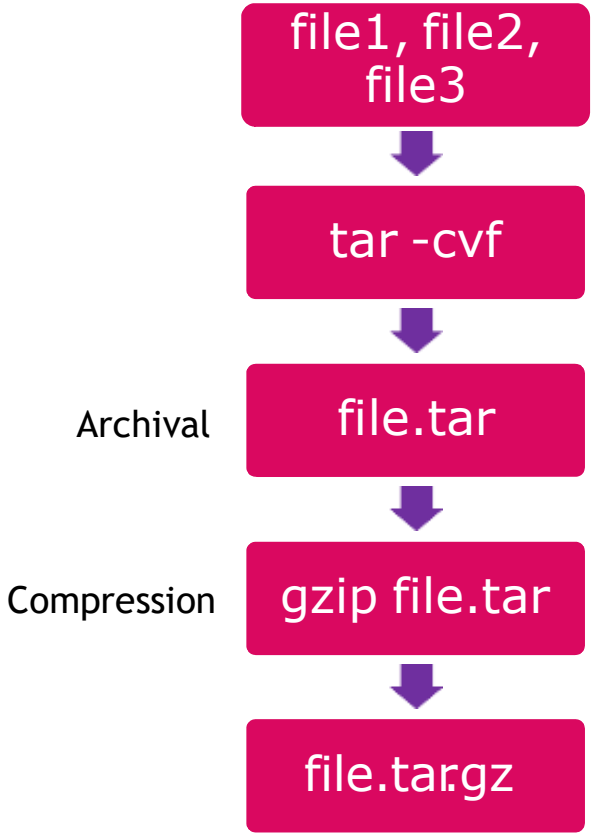
File Archival

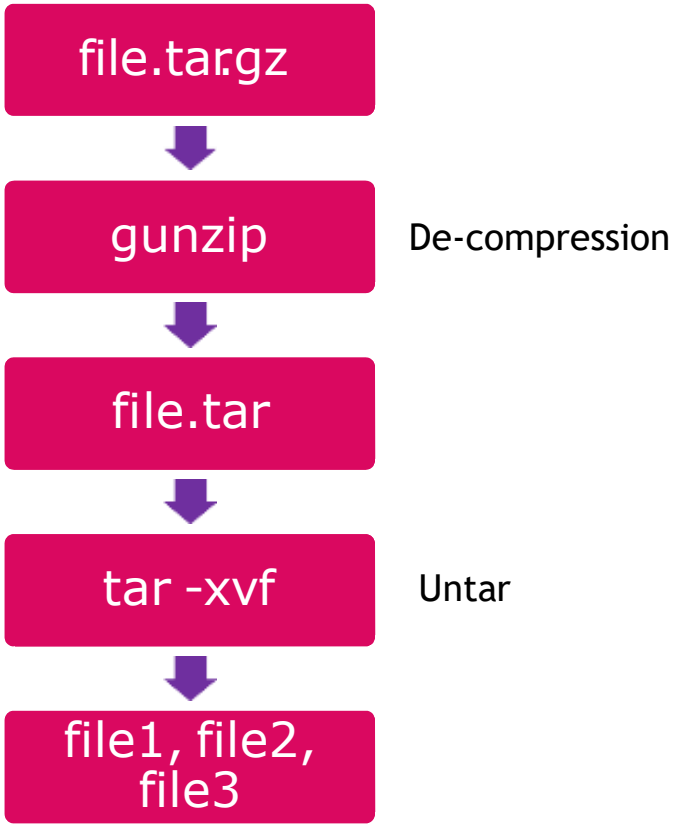
- Used for creating disk archive that contains a group of files or an entire directory structure
- An archive file is a collection of files and directories that are stored in one file
- Archive file is not compressed, it uses the same amount of disk space as all the individual files and directories
- In case of compression, compressed file occupies lesser space
- Combination of archival & compression also can be done

- **File archival is achieved using ‘tar’ with the followingcommands:**
- **tar -cvf <archive name> <file-names>**
- **tar -xvf <archive name>**

Linux Systems

File Archival - Flow





Linux Systems

File Archival - tar

```
user@user:TBD] ls Dir1
Dir2 user@user:TBD] ls
Dir1
1.txt 2.txt 3.txt 4.txt
user@user:TBD] tar cvf dir1.tar Dir1/ # Archive Dir1 as dir1.tar Dir1/
Dir1/1.txt
Dir1/4.txt
Dir1/2.txt
Dir1/3.txt
user@user:TBD] ls
Dir1 dir1.tar Dir2
user@user:TBD] rm -fr Dir1
user@user:TBD] ls dir1.tar
Dir2
user@user:TBD] tar xvf dir1.tar
Dir1/
Dir1/1.txt
Dir1/4.txt
Dir1/2.txt Dir1/3.tx
user@user:TBD] ls
Dir1 dir1.tar Dir2
user@user:TBD]
```

Stay Connected

About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies,

No-1, 9th Cross, 5th Main,
Jayamahal Extension,
Bangalore, Karnataka 560046

T: +91 80 6562 9666
E: training@emertxe.com

https://www.facebook.com/Emertxe

https://twitter.com/EmertxeTweet

https://www.slideshare.net/EmertxeSlides