Linux Notes

CONTENTS

I. INTRODUCTION

- GNU:
  - GNU is a Unix-like operating system
    - it is a collection of many programs: applications, libraries, developer tools, even games
    - GNU is free software that respects users' freedom
  - GNU stands for GNU's Not Unix
  - initiated by *Richard Stallman*
    - also initiated Free Software movement & Emacs
  - GNU C, is referred to as GCC
  - GNU General Public License (GPL)
- Linux:
  - created in 1991, by Linus Torvalds
  - literally, *Linux* is the name of the OS's kernel
  - in popular usage, Linux includes the whole ecosystem
    - including accompanying GNU programs
    - other programs
- Three levels of Linux system:
  1) hardware
  2) kernel
  3) user processes
- Kernel:
  - the *kernel* is the core of OS
  - the kernel is software in memory that tells the CPU what to do
  - the kernel manages HW and is the interface between HW and running program
  - the kernel manages the following:
    - system calls from processes, see next
    - managing processes
    - memory management
    - device drivers
- User processes:
  - *user processes* are the running programs that the kernel manages
  - these processes collectively form the *user space*
- Kernel versus user modes/spaces:
  - the kernel runs in *kernel mode* & has unrestricted access
  - the user processes run in *user modes* which are restricted
  - the part of the main memory that user processors can access is called *user space*
  - the area that only the kernel can access is called the *kernel space*
- Ubuntu:
  - *Ubuntu* is a Linux distribution
  - code name is *xenial*
- User & root user:
  - a *user* is an entity that runs processes & own files
    - users exist to support permissions and boundaries
  - a user is associated with a *username*
  - the kernel identifies users by *userids*
  - every user process has a user *owner*
  - the *root user* is an exception because they can interfere with other users,
    - also known as *superuser*
- Filesystem:
  - Linux has a single filesystem tree
    - this is unlike Windows
    - storage devices are mounted at various points on the tree
  - filenames that begin with . symbol are hidden
  - filenames and commands are case sensitive
  - no concept of file extension
  - do not imbed spaces in filenames
- Permissions:
  - a file's *mode* summarizes the files permission
  - the mode has the following format

    `trwxrwxrwx`
  - t stands for *type*
    - 't=-' means file
    - 't=d' means directory
    - 't=b' means *block*
    - 't=c' means *character*
    - 't=p' means *pipe*
    - 't=s' means *socket*
    - 'b', 'c', 'p', or 's' imply a device
  - the 9 remaining fields are partitioned into 3 sets:
    - permissions for user (u)

    - permission for group (p)
    - permission for other (o)
  - each set has three fields
    - read (r)
    - write (w) and
    - executable (x)
  - absence of a permission is denoted by '-'
  - directories need to be executable to be able to be accessible
- Running an executable file:
  - make sure file permission 'x' is turned on
  - ttype `./fn`
- Common directories:
  - `/bin` contains binaries, or executables, that must be present for the system to boot and run
  - `/boot` contains Linux Kernel, initial RAN disk image (for drivers needed at boot time) and the boot loader
  - `/dev` contains device nodes
  - `/etc`, contains system configuration files
    - shell scripts, password, boot, device and networking setup files
  - `/home`, contain regular user directories
  - `/lib`, abbreviation for library, DLL equivalent
  - `/media`, mount points for removable media such as USB drives and CD-ROMs
  - `/mnt` is a generic mount point under which you mount your file-systems or devices
  - `sys` device and system interface
  - `sbin`, system bin, system executables, for system administrators
  - `/usr` contains bulk of Linux system programs & support files used by users
    - many of the directory names are the same as the root directory, and hold same type of data
    - `/usr/include` holds header files by the C compiler
    - `/usr/info` contains GNU info manuals
    - `/usr/local` is where administrators store their own software
    - `/usr/man` contains manual pages
  - `/var`, variable directory, where runtime information is stored
- Text editors:
  - `emacs`
  - `nano  #basic editor`
  - `vi`
  - `less`
    - see the contents of a file one screen at a time
    - press `spacebar` for next screen
    - press `b` for previous screen
    - press `/text` to search for text forward
    - press `q` to exit
    - can redirect to less `command | less`
- Remote log in:
  - OpenSSH is a freely available version of the *Secure Shell* (SSH) protocol family of tools for remotely controlling, or transferring files between, computers
- X windows system:
  - GUI (Graphic User Interface) uses a terminal emulator to interact with the shell
  - WIMP stands for windows, icons, menus, pointer
  - a *windowing system* (or window system) is a type of GUI which implements the WIMP paradigm for a user interface
    - can think of a windowing system as a device driver
  - the *X Window System* (X11, or shortened to simply X) is a windowing system for bitmap displays, common on UNIX-like computer operating systems
  - think of X as sort of the kernel of the desktop that manages anything from generating windows to configuring displays to handling input devices
    - for example, X windows allows copying using mouse right and middle buttons
  - there is an X *server* and an X *client*
- Desktop environment: Gnome:
  - a *desktop environment* (DE), is a collection of software designed to give functionality and a certain look and feel to an operating system
    - important parts of a DE are the window manage (WM) and the file manager
    - DE provides utilities to set wallpapers and screensavers, display icons on the desktop, and perform some administrative tasks
    - in Ubuntu all DE's uses X windows system
  - Gnome is a desktop environment and is part of GNU Project
    - Gnome3 is the current version
  - Ubuntu tracks Gnome, but uses *Unity shell* rather than Gnome shell

II. BASH

- Shell:
  - a shell is a program that takes keyword commands and passes it to the OS
  - a shell is a *command line interface* (CLI)
  - general command structure is
    `command -options arguments`
  - options could be
    - a single character (e.g. -l) or
    - long option, (e.g. –reverse)
  - a *shell script* are text files that contain a sequence of shell commands
- Shell prompt:
  - commands are entered at the *shell prompt*
  - format is usually user@machinename followed by current working directory
  - $ ending implies a regular user, # implies a superuser
  - a terminal is initialized to its home directory
- Bourne shell:
  - there are many different Unix shells
  - all shells derive features from Bourne shell
  - Bourne shell program is located at */bin/sh*
  - Bourne shell was developed at Bell Labs
- bash:
  - the shell used in Linux/Ubuntu is *bash*
  - bash stands for Bourne-Again Shell
  - bash is an enhanced version of Bourne shell
  - `/bin/sh` is a link to bash `/bin/bash`
- Streams & standard I/O:
  - processes use I/O *streams* to read and write data
  - each process may be associated with
    - *stdin* or *standard input*
    - *stdout* or *standard output*
    - *standard error*
- Redirection & pipe:
  - use the redirection character to send the output of a command to a file

    ```
    command > fn  #fn overwritten
    command >> fn #appended to fn
    ```

  - to send stdout of a command to the stdin another command

    ```
    command1 | command2
    ```

- Control commands:
  - ∧ + Alt + T starts a terminal
  - ∧ + D stops a current standard input
  - ∧ + C terminated a program
- history:
  - up arrow shows previous command
  - `history` command lists history
- Filesystem navigation symbols:
  - the `.` symbol refers to the current working directory
    - when `./` is pre-pended to a filename, then it is searched only in current directory
  - the `..` symbol refers to the parent working directory
  - the ∼ symbol refers to home directory
- Wildcards:
  - wildcards can be used with any command that that accepts fn as argument
  - * symbol represents any characters
  - ? symbol represents a single character
  - [characters] represents any character ∈ set
    - e.g. [abc]
  - [[: class]] represents any character that ∈ class
    - e.g. [[: alnum]], [[: alpha]], [[: digit]], [[: lower]], [[: upper]]
- help with commands:
  - the *manual* command is `man`
    - the manual has eight sections

    ```
    command-name --help #basic help
    man command    #more detailed
    man -k keyword #to search
    info command   #more complex
    ```

- Simple commands:

  ```
  date
  cal   # calender
  clear # brings cursor to top of screen
  exit  # exits terminal
  ```

- Filesystem navigation commands:

  ```
  pwd    # print working directory
  ls     # list
  ls -a  # all files
  ls -l  # long format
  ls -t  # sorted- last modified time
  cd     # change directory
  cd     # to home directory
  ```

- File & directory manipulation:

  ```
  cp             # copy
  cp item1 item2 # 1->2
  cp item1 item2 dir #1,2 -> d
  cp -R ...      # recursive, with dir
  mv             # move
  mkdir          # make directory
  rmdir          # remove directory
  rm -rf * #recursively remove files/dir
  rm             # remove
  touch fn       # creates fn
  file fn        # tells type of fn
  diff fn1 fn2
  ```

- `echo text`
  - echo prints its argument to stdout
  - `echo $X` prints value of X variable
- `cat fn`
  - cat reads files sequentially, writing them to stdout
  - the name cat is derived from its function to concatenate files

- `grep text dir_fn`
  - looks for a text in directory or filename dir_fn
  - -i option makes text case-insensitive
- `find dir -name fn -print`
  - to find fn in directory dir
- `passwd`
  - to change password
- Processes:
  - a process can be executed in background by ending the command with `&`
  - each process on the system is has a numeric *process ID*, or PID
  - `ps` reports a snapshot of the current processes

```
ps -elf #e=every, f=full-format
ps x    #all
ps -elf | grep lightdm
kill pid #terminates, last resort
```

- Changing file permission: `chmod`

```
chmod g+r fn #read permission to group
chmod o-w fn #removes write permission
chmod 644 fn #octal, absolute change
```

- Link & alias: `ln` & `alias`
  - a link can be hard (`ln`), or symbolic (`ln -s`)
  - a *symbolic link* is a file that points to another file or directory effectively creating an alias
  - symbolic links offer quick access to obscure directory paths
  - to create a symbolic link, type

```
ln -s target link-name
```

    where the `target` is where the link is pointing to,
  - `alias` lists the current aliases
- Compress and archive:

```
gzip fn  #compress, 1 fn, --> fn.gz
gunzip fn.gz  #fn.gz --> fn
tar cvf archive.tar fn1 fn2 #fn->ar
tar xvf archive.tar     #archive->fn
tar tvf archive.tar #table-of-content
```

  - .tar suffix is convention nor requirement
  - c flag means create
  - x flag means extract
  - f flag should precede archive file
  - p flag preserves permissions
  - an archive could be compressed `fn.tar.gz`
- `zip` & `unzip`:
  - to `zip` all files in a directory type

```
zip fn *
```

  - `unzip fn`
  - `fn` does not not include extension
- `ldd`
  - `ldd` invokes the standard dynamic linker
  - the library GLIBC is the standard dynamic linker

III. Environment & Configuration

- Shell variable:
  - the shell can store temporary variables, called *shell variables*,

```
Y=10   #to set, no space
Y="$Z" #assignment
echo $Y
```

- Environment variable & `export`
  - an *environment variable* is similar to a shell variable but it is not specific to a shell,
  - the OS passes the environment variables to programs that the shell runs,
  - assign an environment variable using the `export` command,
  - exporting a variable makes the variable available to all sub-shells and processes created by that shell,
  - it does not make it available everywhere in the system, only by processes created from that shell.
- `set`
  - to set or unset options and positional parameters,
  - without options displays both the shell and environment variables.
- `printenv`
  - displays only environment variables,
  - `printenv Y` lists the value of Y.

A. Environment variables

- `PATH`
  - a *command path* is a list of system directories that the shell searches when trying to locate a command,
  - `PATH` is a special environment variable that contains the command path,
  - can add a directory `dir` first or last as follows

```
PATH=dir:$PATH
PATH=$PATH:dir
```

- `LD_LIBRARY_PATH`
  - dynamic link library path.
- `DISPLAY`
  - X window system variable,
  - is the name of the display if running a graphical environment,
  - general format is

```
DISPLAY = hostname:D.S
```

    * omitted hostname means local host,
    * D means display number,
    * S means screen number.
  - for example, `:0` means its the first display generated by X server,
- `TERM`
  - name of terminal type
  - `xterm`.
- `HOME`
  - pathname of home directory,

  - `/home/ara`.
- `USER`
  - username,
  - `ara`.
- `PS1`
  - stands for *prompt string* 1,
  - built-in shell variable,
  - defines the contents of the shell prompt,
  - initially defined in `/etc/profile`,
  - then modified in `/home/ara/.bashrc`.
- `SHELL`
  - name of shell program being used,
  - `/bin/bash`.

B. *Configuration files*

- Startup files:
  - when a user logs on, the bash program initiates a series of configuration scripts called *startup files*,
  - there are global scripts followed by user scripts,
    - global startup files in `/etc` directory, define the default environment shared by all users,
    - user startup files, `/home/ara`, can be used to extend or override settings in global configuration script,
  - in addition, there are two kinds of shell sessions: login shell sessions & non-login shell sessions,
    - a *login session* prompts for user name & password,
    - a *non-login session* typically occurs when a new terminal is launched.
- `/etc/profile`
  - global startup file for login sessions,
  - in interactive session it launches `/etc/bashrc`,
  - also execute the scripts `/etc/profile.d/*.sh`.
- `/home/ara/.profile`
  - user startup file for login sessions,
  - in interactive session it launches `/home/ara/.bashrc`,
- `/etc/bash.bashrc`
  - global startup file for non-login sessions,
  - is used to set environmental items for a users shell,
  - is executed for both interactive and non-interactive shells,
- `/home/ara/.bashrc`
  - user startup file for non-login sessions,
  - I have CUDA initialization, and my aliases.
- `/etc/X11/xorg.conf`
  - the X configuration file provides a means to configure the X server,
  - the NVIDIA driver includes a utility called `nvidia-xconfig`, which is designed to make editing the X configuration file easy,
  - for diagnosis read `/var/log/Xorg.0.log`
    - lines that start with (EE) indicate error.
- Activating changes: `source`
  - changes to startup files will not take effect until the terminal session is closed and start a new one,
  - `source` is a bash shell built-in command that executes the content of the file passed as argument, `source fn`
  - `source` has a synonym in the symbol '.' (dot) `. fn`

IV. System Administration

- Version I am using:
  - Ubuntu version 16.04.1 LTS
  - kernel version $4.4.0-47$−generic
  - GCC 5.4.0
  - GLIBC 2.23
  - Python 2.7.12
- `lsb_release -a`
  - lsb means Linux Standard Base
  - tells you about OS details
- `uname`:

```
uname -m #32 vs 64 bit
uname -r #kernel version
```

- `sudo`
  - super user, administrative rights
  - precede any command with sudo
  - can enter super user mode by typing

```
sudo su
```

- Storage management:

```
df -h  # disk free, h=human
du -h  # disk (dir) usage
free   # free memory
```

- Dependency clashing:
  - if a package requires a shared resource (shared library), it is said to have a *dependency*
  - dependencies can be problematic when two different applications require incompatible versions of the same dependency
- Dependency environments:
  - to address dependency clashing, some package distribution managers have software that create environments, inside of which specific versions of software can be maintained independent of those contained in other environments
- Operating-system-level virtualization:
  - *OS-level virtualization* is a server virtualization method in which the kernel of an OS allows the existence of multiple isolated user-space instances, instead of just one
  - such instances are sometimes called *containers*, *software containers*, *virtualization engines* or *jails*
  - containers may look and feel like a real server from the point of view of its owners and users
  - *Docker* is an open-source project that automates the deployment of applications inside software containers
  - recommended to use TensorFlow in a container if running over multiple computers
- `dmesg`
  - `dmesg` stands for *display message*
  - `dmesg` is used to examine or control the kernel ring buffer
    - `dmesg` lists the message buffer of the kernel
  - the output typically contains the messages produced by the device drivers
  - when the computer fails unexpectedly after reboot can use this command
- Boot
  - to change boot parameters check `etc/default/grub`

A. Package Management

- Package management & distributors:
  - *package management* is a method of installing and maintaining software on the system
  - one of the most important factors of a distribution is the quality of the packaging system
  - most distributions fall into one of two camps of packaging technologies:
    - *Debian* style (.deb) that includes Ubuntu
    - Red Hat style (.rpm)
- Package file:
  - the basic unit of software in a packaging system is the *package file*
  - a package file is a compressed collection of files that comprise the software package
  - the package file may include programs, data files, metadata, pre- and post-installation scripts, etc.
- Repositories:
  - most packages today are created by the distribution vendors
  - a distribution makes packages available to the user in central *repositories*
  - a distribution may also have related third-party repositories
  - package management system provide some method of dependency resolution
- Ubuntu package manager:
  - `dpkg` is a low level manager
  - `apt`, or *advanced package tool*, is high-level manager
    - also stands for *aptitude*?
- `dpkg`

```
dpkg -l #lists installed packages
dpkg -i package #install
dpkg -r package #remove
dpkg --status package, #installed?
dpkg --search fn #package for fn
```

- `apt`

```
apt-get update #updates list only
apt-get upgrade #does installation
apt-get install xyz
apt-get remove package
apt autoremove #removes unnecessary
apt-cache search search-string
apt-cache show package
apt list --installed #installed packages
```

- PPA
  - a *personal package archive*, or PPA, is a software repository for uploading source packages to be built and published as an `apt` repository
  - PPA offers stable proprietary Nvidia graphics driver updates, without updating other libraries to unstable versions
  - to add graphics driver PPA:

```
sudo add-apt-repository
    ppa:graphics-drivers/ppa
```

B. Devices

- Device drivers:
  - device drivers are part of the Linux kernel
    - not in repository
    - usually in `/dev` directory
    - base path for devices is `/sys/devices`
  - the kernel presents many of the I/O device interfaces to user processes as files
  - exceptions when the driver is not in kernel:
    - device is too new
    - device is exotic, can compile
    - HW vendor is hiding something
- `ubuntu-drivers devices`
  - Ubuntu specific command that lists devices
- `lshw`
  - `lshw` stands for *list hardware*
  - `lshw` extracts information about the hardware configuration of the machine

```
lshw -class display
lshw -short   #lists H
```

- `lspci`
  - `lspci` displays information about PCI buses in the system, and about the devices connected to them

```
lspci -v | grep -i nvidia
sudo update-pciids #update PCIe dev
```

- Display manager: LightDM
  - a *display manager* is responsible for starting the display server and loading the desktop after you type in your username and password
    - display manager is not the same thing as a window manager or a display server
  - *LightDM* stands for *lightweight display manager*
  - LightDM is the default display manager for Ubuntu
    - LightDM is an X display manager
    - it starts the X servers, user sessions and greeter
  - the login window is known as the *greeter*
    - LightDM offers separate greeter packages
    - the default greeter in Ubuntu is *Unity Greeter*
  - display manager `lightdm` should be temporarily turned off before making changes to this config file and then restored
- `service`
  - can use `service` stop and start services temporarily

```
service --status-all
service lightdm status
sudo service lightdm stop
sudo service lightdm start
sudo service lightdm restart
```

- `update-alternatives`
  - creates, removes, maintains and displays information about the symbolic links,
  - `--config fn` shows available alternatives for a link group and allow the user to interactively select which one to use,
  - monitor showed the right resolution after I entered

```
sudo update-alternatives --config
    x86_64-linux-gnu_gl_conf #debug
```

C. C & Python

- GCC
  - GNU compiler for C++,
  - to find version type `gcc -v`.
- GLIBC
  - the GNU C Library is the GNU Project's implementation of the C standard library,
  - despite its name, it now also directly supports C++,
  - to find version type `ldd --version`.
- Python:
  - start a Python session by typing `python`
  - to find version type `python --version`
  - in `/usr/local/lib/python2.7`
- `python-pip`
  - `pip` stands for *Installs Python* (recursive)
  - `pip` is a package management system to install and manage software packages written in Python
  - `apt-get install python-pip`
  - `python -m pip install --upgrade pip`
  - `pip install --upgrade pip`
  - `pip install --user scipy`
  - `pip install --user matplotlib`
  - `pip install --user sympy`
  - can use `pip` to install basic TensorFlow
- `python-dev`
  - `python-dev` is the package that contains the header files for the Python C API
  - `python-dev` is used by lxml because it includes Python C extensions for high performance
  - `apt-get install python-dev`
- `python-numpy`

```
apt-get install python-numpy
pip install --user numpy
```

- `python-wheel`
  - wheels are the new standard of python distribution
  - support is offered in pip ≥ 1.4
  - a wheel is a ZIP-format archive with a specially formatted filename and the .whl extension
  - wheel is designed to contain all the files that is very close to the on-disk format
  - `apt-get install python-wheel`
- Anaconda:
  - *Anaconda* is an open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment
  - `https://www.continuum.io/why-anaconda`
  - I have used Anaconda as a Python compiler
  - both Python versions are included in Anaconda
  - *Conda* is Anaconda's package manager application that quickly installs, runs, and updates packages and their dependencies
    - Conda is also an environment manager application
- PyCharm
  - *PyCharm* is an Integrated Development Environment (IDE) used for programming in Python,
  - `https://www.jetbrains.com/pycharm/`
  - in Windows, used Anaconda compiler in PyCharm.
- Jupyter Notebook:
  - formerly *iPython Notebook*
  - interactively write documents that include code, text, output and *LateX*
- Python environments to address dependency clashing:
  - recommended to use a Python dependency environment when running TensorFlow on a single computer
  - two possibilities include:
    - for the standard distributions use `Virtualenv`
    - Anaconda comes with a built-in environment system
- `python-virtualenv`
  - to create a virtual Python environment
  - `apt-get install python-virtualenv`
  - create a directory, e.g. `env`, that contains this environment
  - create environment using `virtualenv` command

```
virtualenv --system-site-packages
    ~/env/tensorflow
```

  - activate environment using `source` command

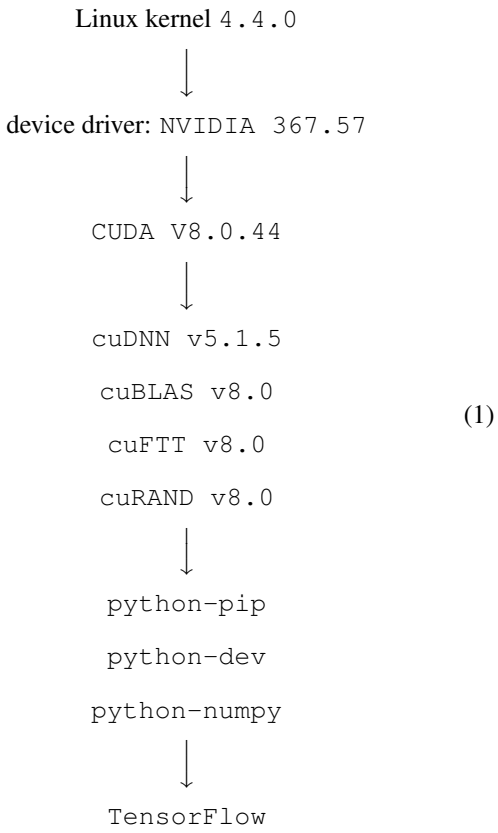
```
source  ~/env/tensorflow/bin/activate
```

  - environment should be active when installing with `pip`
  - exit environment by the command

```
deactivate
```

D. Accelerated computing

- Installation hierarchy:



(1)

- NVIDIA GPU cards & drivers:
  - Titan X (Pascal), GeForce 10 series, GP102
  - GeForce GT 710B, GK208
  - installed in `/usr/lib/nvidia-367`
  - `nvidia-smi` shows GPUs and driver
  - driver version can be founds as `cat /proc/driver/nvidia/version`
  - `nvidia-xconfig` configures xorg.conf
- `CUDA` installation:
  - download from `https://developer.nvidia.com/cuda-downloads`
    - use `deb(local)` option which is a large package
  - to install downloaded package

```
sudo dpkg -i fn.deb
sudo apt-get update
sudo apt-get install cuda
```

  - `libcupti-dev`
    - the `libcupti-dev` library is the NVIDIA CUDA Profile Tools Interface
    - this library provides advanced profiling support
    - to install type `sudo apt-get install libcupti-dev`
  - `nvcc`
    - the `nvcc` command runs the compiler driver that compiles CUDA programs
    - `nvcc` calls the GCC compiler for C code, & the NVIDIA PTX compiler for the CUDA code
    - `nvcc --version` prints CUDA version
- `CUDA`
  - installed in `/usr/local/cuda`
    - `CUDA_HOME=/usr/local/cuda-8.0`
  - `CUDA_VISIBLE_DEVICES`
    - NVIDIA environment variable that specifies PCI device 0 (Titan) to be a CUDA device
    - defined it in `/home/ara/.bashrc`
  - `cudaSetDevice(a)` sets GPU a for CUDA computing
  - `cuda_devices` is an alias that shows active CUDA devices
  - after installation, query CUDA device by typing `/usr/local/cuda-8.0/samples/bin/...x86_64/linux/release/deviceQuery`
- `cuDNN`
  - separate add-on designed for DNN
  - download from `https://developer.nvidia.com/cudnn`
  - inside `/usr/local/cuda` directory type

```
tar xvzf fn.tgz
cp cuda/include/cudnn.h include
chmod a+r include/cudnn.h
cp cuda/lib64/libcudnn* lib64
chmod a+r lib64/libcudnn*
```

- `tensorflow`
  - TensorFlow is open source code provided by Google
  - if using GPU, then build TensorFlow from source
  - from home directory, clone latest TensorFlow source code from GitHub

```
git clone --recursive-submodules
    https://github.com/
    tensorflow/tensorflow
cd tensorflow
./configure
```

  - Bazel
    - TensorFlow uses *Bazel* to build executable code from source code
    - Bazel is open source code provided by Google
      - Bazel is similar to *make*
    - install Bazel
    - create TensorFlow executable

```
bazel build -c opt --config=cuda
    //tensorflow/tools/pip_package:
    build_pip_package
```

  - Python packages TensorFlow
    - dependency clashing prevention is done through Python