**linux**

**基础知识：**

ifconfig -a：查看IP和MAC地址，默认eth0中inet后为IP，ether后为MAC地址。Windows：ipconfig/all

linux中文件存储是离散的，移动文件时只移动指针不移动数据位置

电脑上传到linux系统用压缩文件会更快，压缩后文件的内存存储是连续的

命令前加sudo表示用管理员权限运行

本地电脑可以通过[ssh 服务器公网IP]+输入账号密码连接正常的服务器，账号一般为root，root账号有管理员权限，子账户的限制比较多

正常的服务器既可以访问外网，也可以从外网访问服务器。但很多服务器会对从外部访问服务器做限制，只能通过内部网络访问，保密性高的服务器甚至不能访问外网

**快捷键：**

CTRL+C：取消

CTRL+Z：把运行程序放后台运行

CTRL+SHIFT+C：复制

CTRL+SHIFT+V：粘贴

**基础命令(文件的意思包括单个文件和目录)：**

按↑↓键可以快速显示之前输入过的命令

ls/ll：查看当前目录下的文件

ls |head -n 100：显示当前目录中前100个文件

df -h：linux查看磁盘空间

du -h --max-depth=1：查看当前目录下的所有文件大小

TAB：可以快速选择存在的路径

cd 文件：进入当前目录下的文件

cd ..：返回上级目录

cd /：返回根目录

cp -r 待复制的文件 目标目录：复制文件(以及子文件)到目标目录下

mv 文件名 新名字：更改文件名

rm -rf 文件：强制删除整个文件

mkdir 文件夹：创建文件夹

cat 文件：查看文件的详细信息

ln 文件 镜像文件：为文件创建一个镜像文件，镜像文件会占用空间

ln -s 文件 镜像文件：为文件创建一个镜像文件，镜像文件不占空间

whereis 文件名：搜索文件

nvidia-smi：查看GPU使用情况

watch -n 0.1 nvidia-smi：实时显示GPU使用情况。CTRL+C退出

fuser -v /dev/nvidia\*：查看显存中隐藏运行的程序

kill -9 程序编号：停止正在运行的程序

find ./ -name 文件名：查找当前目录下所有的该文件

reboot：重启服务器

lspci | grep -i nvidia：查看显卡型号

ls -l|grep "^-"| wc -l：当前目录中非文件夹文件数量(不包括子文件夹)

ls -lR|grep "^-"| wc -l：当前目录中非文件夹文件数量(包括子文件夹)

curl www.baidu.com：访问网址得到HTML

sudo fuser -v /dev/nvidia\* |awk '{for(i=1;i<=NF;i++)print "kill -9 " $i;}' | sudo sh：清除所有gpu占用。非正常退出时，nvidia-smi不会显示一些残留的程序

**tmux：**

tmux ls：查看所有存在的窗口(要在主页面使用)

tmux new -s 窗口名：新建窗口。窗口中的程序只要不暂停会一直在服务器后台运行，即使关闭操作的页面。窗口中的程序和主页面中程序的可以同时运行

CTRL+B+D：从窗口返回主页面(窗口仍在)

tmux a -t 窗口名：从主页面进入一个窗口

tmux kill-window -t 窗口名：删除一个窗口

**sh：**

sh .sh安装包：安装.sh安装包

sh：进入sh命令页面。不进入也可以使用sh安装

CTRL+D：退出sh页面

**zip：**

zip文件是无损压缩，压缩前后大小几乎不变，相比其他压缩格式占内存大

unzip .zip：解压.zip压缩包

**tar：**

tar -xvf .tar压缩包：解压.tar压缩包

**权限管理：**

sudo chmod -R 777 文件：赋予文件管理员权限

Ll：查看当前目录下的所有文件和文件权限，上传到服务器的文件如果没有权限可以在命令行中复制一份，复制后的有权限，但需要通过命令行删除

**环境变量：重启后会复原**

echo $PATH：查看当前PATH环境变量。环境变量间以:隔开

export PATH=$PATH:文件路径：将文件添加到环境变量中。$PATH表示保留有的环境变量，:表示变量间的分隔符号

export PATH=多个文件路径：会更改所有环境变量

echo $LD\_LIBRARY\_PATH：查看当前动态链接库环境变量

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:动态链接.so库：添加.so库

**动态库环境变量LD：**

ld --verbose | grep SEARCH：查看所有ld路径

sudo ldconfig 动态库.dll所在目录：添加动态库到环境中(重启依然生效)

**服务器：**

服务器有主机地址、内网地址、公网地址，分别包括IP、掩码、广播地址。服务器之间连接要知道对方的主机IP地址，从互联网上接入要知道公网地址

密码连接服务器：设置好服务器地址、端口(22)、用户名(root)，通过密码连接

公钥连接服务器：服务器接受电脑的公钥，设置好服务器地址、端口(22)、用户名(ubuntu)，电脑就可以通过私钥连接上服务器，一台服务器也可以通过同样的方法连接到其他服务器

ifconfig：查看服务器的网络接口(IP、掩码、广播地址)。eth0为外网地址，lo为内网地址

ping 服务器ip：连接进入服务器

**NFS服务器挂载：**

有的服务器只能通过密钥登录的挂载，不能通过密码登陆的挂载

服务器端的共享目录要设置允许其他人访问的权限

sudo apt install nfs-kernel-server：安装nfs工具

sudo mkdir 文件夹： 创建挂载数据文件夹

sudo chmod -R 777 文件：赋予文件管理员权限

sudo mount -t nfs 服务端ip:服务端挂载文件路径 挂载到客户端路径 -o nolock：从服务端挂载文件到客户端(重启客户管会消失)

umount 挂载到客户端路径：取消挂载

**rsync数据同步：**

可以同步同一服务器的两个文件，也可以同步不同服务器上的文件。同步不同服务器上文件时会11使用ssh连接被同步的服务器，被同步的服务器不需要安装rsync

rsync -av [文件1] [文件2]：将文件1同步到文件2目录中(可以使用’文件1/’)

rsync -av [本机文件路径] [用户名]@[目标主机ip]:[目标主机文件路径]：同步数据到目标服务器

**显卡驱动：**

sudo apt-get --purge remove nvidia-\*：卸载原有的显卡驱动

**gcc：Linux编译工具**

gcc -v：查看版本

gcc下载地址：<https://mirrors.aliyun.com/gnu/gcc/>

选择合适的gcc版本，并下载tar.gz

**python：**

linux中python的os.listdir()函数取出的路径排序是随机的

**Git**

分布式版本控制工具。由Linux的创造者Linus所写，版本控制用于记录文件修改历史，方便在不同版本进行切换修改，便于团队协作

git分为工作区(写代码)、暂存区(临时存储)、本地库(历史版本)。每个文件夹/项目相当于一个本地库

在Git Bash中输入命令执行功能，Git Bash中会显示当前所在操作的目录路径。右键任意文件目录打开Git Bash则让Git直接进入到当前目录

分支思想：分支相当于备份，在备份中开发代码不影响主程序功能的使用，最后再把开发好的代码合入主程序中。方便同时进行多个项目开发

**Git命令**

**设置：**

用户名和邮箱名只需设置一次，不需要真实存在。设置后在C盘用户文件夹下会生成.gitconfig文件记录信息，不设置无法提交代码

git config --global user.name 用户名：设置用户名

git config --global user.email 邮箱名：设置用户邮箱名

**本地操作：**

git init：初始化本地库，让git获取当前目录的管理权限(会显示分支名)

git status：查看本地库状态。只在工作区的文件为红色，添加到暂存区的文件为绿色，已经提交到本地库的文件不显示

git add 文件名：将文件添加到暂存区(当文件名为.时为当前整个目录名)

git commit -m “日志信息” 文件名：将暂存区文件提交到本地库

git reflog：查看历史记录，有简略版本号和日志信息

git log：更详细的版本号和更多信息

git reset -mixed 版本号：回退指定版本，但不修改当前文件，不加版本号某认上一个

git reset --soft：撤销commit操作

git reset --hard 版本号：版本切换，会清除当前所有的修改，不加版本号某认上一个

**分支：**

git branch 分支名：创建分支

git branch -v：查看分支。带\*为当前所在分支

git checkout 分支名：切换分支

git checkout -b 分支名：切换分支，如果没有该分支则从当前分支复制创建新分支

git merge 分支名：把指定的分支合并到当前分支上

git branch --delete 分支名：删除本地仓库的分支

git push origin --delete 分支名：删除远程仓库的分支

**远程操作：**

在github/gitlab中有HTTPS链接和SSH链接，HTTPS链接是公开的，一般用于github上公开项目的分享，使用时需要登陆密码，只能从网站上下载代码，但不能修改代码上传网站。SSH链接只有管理员和所有者可以使用，可以修改代码上传网站，需要设置密钥，一般公司的私人仓库只用SSH链接

ssh-keygen -t rsa -C 用户邮箱：生成ssh私钥和公钥。连按三个回车后生成密钥在用户目录下的.ssh中，将公钥复制粘贴到github/gitlab网站自己账户下的ssh公钥存放处，就可以对自己管理的项目可以使用ssh链接

git remote -v：查看所有建立的链接

git remote add 别名 远程库链接：用别名(一般为origin)替代链接

git remote rename 别名1 别名2：更改别名1为别名2

git remote rm 别名：删除别名

git config --global http.sslVerify "false"：第一次克隆时如果显示证书不合格，使用此设置

git clone 远程库链接(不能用别名)：将远程库复制到本地库，同时会自动创建别名origin建立链接(如果已经创立相同链接的别名则不动)。一般本地文件夹/项目名要和远程库中项目名一致

git clone -b 分支名 远程库链接：复制远程库分支到本地

git push 别名 本地库分支：将本地库分支更新到远程库(需要登陆，如果远程库是他人的，需要在github上加入远程库管理团队)

git pull 别名 远程库分支：将远程库分支更新到本地库(第一次用clone)

**.gitignore**：

可以在项目中创建.gitignore文件写入不需要推送到仓库的文件

\*.jpg：不会推送以.jpg结尾的文件

file：不会推送所有名称为file的文件夹/文件

file/：不会推送所有名称为file的文件夹

!文件名：会推送该文件

**git-lfs：大文件下载**

sudo apt-get install git-lfs：linux安装git-lfs。windows安装git时自带

git lfs install：启用git-lfs，可以下载大文件

git lfs uninstall：关闭git-lfs

**Github**

基于git的代码库托管站(国外网站)，某种程度上就是代码的网盘，也有开源开发商作为代码发布、存放、征集Bug和意见用

**README.md：**

README大写：大写字母的ascall码比较小，用大写使README的排序靠前

md文件兼容html代码

每行前面加#/##/###/...显示为一级标题、二级标题...

段落结尾加两个空格实现换行

\*内容\*：斜体\*\*内容\*\*：粗体。\*\*\*内容\*\*\*：斜粗体

&#10084;：爱心符号

&#9733;：五角星符号

&#10003;：打勾符号

\*\*\*：分割线

>/>>/>>>：缩进

```内容```：加修饰框，一般用于写代码。里面的内容不需要加换行符，会自动换行

![image](README\_IMAGE/001.jpg)：显示图片

**Docker**

基于go语言的开源容器引擎，可以将代码和环境放到一个可移植的容器中，方便部署到其他的linux机器上。docker容器轻量化，内存需求极低

镜像：包含程序运行环境的压缩包。包含操作系统、cuda版本、python版本、库等，但不包含cuda驱动等底层东西。一台新的机器只需要cuda驱动等满足条件，就可以加载镜像来创建程序运行环境

容器：使用镜像后创建的程序运行环境。即使在同一个服务器上，各容器之间也是互不干扰的，每个容器都有自己独立的空间，包括文件系统、内存、磁盘、进程、库和环境等

仓库：存放镜像的地方。仓库的管理方法和git很相似

容器与虚拟机：容器和虚拟机的目的都是将一个服务器划分为互不干扰多个区域。虚拟机划分的更彻底，包括操作系统；容器共享操作系统，只包含库等环境。容器的创建速度和资源消耗比虚拟机小很多，容器启动速度可达秒级，因此通常情况下只会使用容器

docker镜像保存在本地的var文件中

sudo apt-get install docker：linux中下载docker

systemctl restart docker：重启docker

docker version：查看docker版本

docker pull 镜像源：下载镜像

docker images：查看已有镜像

docker save -o 保存名称.tar 镜像名/id：保存镜像为.tar文件。id可以不是完整的

docker load -i .tar包：加载镜像。加载后的镜像没有repository和tag

docker tag 镜像id repository:tag：给镜像打标签，id可以不是完整的。repository要按照镜像仓库的格式，tag一般为版本号

docker push repository:tag：上传镜像至镜像仓库。需要事先连接镜像仓库

docker ps -a：查看所有存在的容器(包括正在运行和停止运行的)

docker run 镜像名/id：创建并运行容器。会给出随机的容器名/id

docker run -p XX:xx -d --name 容器名 镜像名/id：-p设置端口映射，XX为服务器端口，xx为容器端口，之后访问容器通过服务器的XX端口；-d为后台启动；name后为容器名

docker stop 容器名/id：停止容器

docker start 容器名/id：启动容器

docker restart 容器名/id：重启容器

docker logs 容器名/id：查看容器日志

docker rm 容器名/id：删除容器。加上-rf可以强制删除正在运行的容器

docker exec -it 容器名/id /bin/bash：进入容器

docker exec -it 容器ID /bin/bash：进入容器环境

exit：退出容器环境

**Dockerfile：**

Dockerfile：直接创建镜像的配置文件，构建镜像时会运行文件中的命令

FROM 其他镜像：加载其他镜像，以其他镜像为模板。此行必须要有且作为第一个

WORKDIR 路径：进入某个路径中，之后的操作也会在这个路径下执行

RUN 命令1 && 命令2：构建镜像时会执行的命令。RUN指令会产生缓存以便下次创建，使用--no-cache不保存缓存。如果一行太长写不下时，加入’ \’换行。可以把命令放到一个RUN后，也可以拆分为多个，多个时会略微增加大小

CMD 命令：构建容器后、启动容器时会执行的命令

**Kubernetes**

Kubernetes(k8s)是2014年google开源的容器编排调度引擎，用于管理多个服务器上容器的部署、管理和扩展。基于golang语言实现

k8s集群：由控制面板、节点、其他组件组成

控制面板：用于管理整个集群的主服务器，所有对集群的操作都要经过控制面板

节点：单个服务器或虚拟机，每个节点都有独立的操作系统，集群中至少包含1个节点

附加组件：基于k8s制作的应用

pod：最小可部署单元，包含1个或多个容器

容器管理：容器删除时IP、文件等信息也会消失，但k8s可以记录这些信息以便重启。当容器因为运行或内存等出错时，k8s可以快速删除并重启。可以通过k8s自定义容器的复制、更新、启动、删除等复杂功能，实现系统的更新或回退

配置管理：服务器配置、线程池参数、账号密码等可以通过k8s统一管理

工作流程：[可视化界面UI/命令行CLI] -> [API] -> [控制面板] -> [节点]

控制面板：

kubecti工具：通过控制面板所在服务器的linux命令行操作来控制集群

dashboard可视化界面：官方的可视化界面

API接口(API serber)：管理API功能

调度器(scheduler)：不同节点的规格可能不一样，调度器会根据算法将pod放到更合适的节点上

管理控制器(controller)：包括节点、任务、端点分片、服务账号控制器，可以外接第三方云平台

数据库(etcd)：键值存储类型的数据库

节点：

kubelet：管理pod、节点存储、网络

kube-proxy：网络代理，4层负载均衡，内部服务发现

container-runtime：容器运行时的环境。比如docker等