28. multitail：实时监视文件（tail++）
    1. Installing on Ubuntu
    2. Usage
29. mycli
30. ncdu：磁盘空间管理利器
    1. Installing on Ubuntu
    2. Usage
31. neofetch/screenfetch：看起来很厉害的系统信息查看器
    1. Installing on Ubuntu
    2. Usage
32. node
33. nvm
34. pstree
35. pv
36. ripgrep
37. shellcheck
38. sl
39. sonarqube-lts
40. starship
41. thefuck
42. tig
43. tldr
44. tmux
45. tree
46. wrk
47. ranger
48. you-get
49. zoxide

CS:APP 环境搭建（以及顺便做的一些事情）

- CS:APP 环境搭建（以及顺便做的一些事情）
  - 虚拟机
    - Parallels Desktop
    - VMware Fusion
    - Multipass

        - Usage
    - neofetch/screenfetch：看起来很厉害的系统信息查看器
        - Installing on Ubuntu
        - Usage
    - node
    - nvm
    - pstree
    - pv
    - ripgrep
    - shellcheck
    - sl
    - sonarqube-lts
    - starship
    - thefuck
    - tig
    - tldr
    - tmux
    - tree
    - wrk
    - ranger
    - you-get
    - zoxide

在 CS:APP 中文版的序言中曾经说道：

> 本书采用以下组合方式：在经典的 x86-84 架构机器上运行 Linux 操作系统，采用 C 语言编程。

但是，众所周知的是，M1 Mac 基于 ARM 架构，而且，虽说 MacOS X 是典型的类 Unix 操作系统，但他毕竟不是 Linux。操作系统和架构的差异势必使环境搭建的问题变得有些麻烦。

想要在 M1 Mac 上跑 Linux/amd64 ，比较直观的思路有两种：虚拟机和 Docker。

虚拟机

当前适用于 M1 Mac 的（我了解的）虚拟机大致有三种，Parallels Desktop，VMware 和 Multipass。

Parallels Desktop

Parallels Desktop 成熟且强大，点击官网下载开盒即用。

代价是什么？永久授权698RMB，但不支持大版本间的免费升级（小版本还是支持的）；订阅498RMB，支持免费升级，但只能用1年。

为了捍卫 CS 开源免费的精神，我最终没有采用此方案（在浪费了14天的试用期后）。

VMware Fusion

VMware 理论上是支持 M1 Mac 的，但总感觉不太好用（我认为）。譬如 VMware 本身并没有支持 Ubuntu 的图形化界面，又譬如我执行到最后莫名其妙地会卡住。

具体流程可以参考此教程。

Multipass

Multipass 是 Ubuntu 官方推出的虚拟机应用，可以通过Ubuntu官网下载，也可以通过 Homebrew 得到。具体使用可以参考此教程和此教程。理论上来讲，Multipass 应该比 Docker 更简洁高效，但是 Docker 看起来有种纯真的美不是吗?

此处简略介绍一下 Multipass 的使用方法。

1. 使用 homebrew 安装 multipass

   ```
   brew install multipass
   ```

2. 使用 multipass 新建虚拟机

   ```
   multipass launch -n vm -c 4 -m 8G -d 64G
   ```

   - -n：指定虚拟机名称，此处为 vm。
   - -c：指定 cpu 核心数，此处为4。
   - -m：指定内存，此处为8G。
   - -d：指定占用硬盘大小，此处为64G，建议至少20G。

3. 删除虚拟机

```
#只是将vm移入回收站中，并没有彻底删除。
multipass delete vm
#将vm移出回收站
multipass recover vm
#清空回收站，将vm从回收站中移除，彻底删除。
multipass purge
```

4. 运行虚拟机

```
#以命令行形式运行虚拟机vm(进入虚拟机)
multipass shell vm
#让虚拟机运行lsb_release -a命令查看实例基本信息
multipass exec vm --lsb_release -a
```

5. 其他操作

```
#列出所有虚拟机
multipass list
#查看vm的详细信息
multipass info vm
#停止虚拟机
multipass stop vm
```

6. 安装 Ubuntu 的 GUI 桌面

```
#更新apt源
sudo apt-get update
#下载Ubuntu Desktop
sudo apt-get install ubuntu-desktop xrdp -y
#创建新用户并填写相应信息
sudo adduser username
#为用户赋予sudo访问权限
sudo usermod -aG sudo username
#查看vm的ip信息并复制之(也可以通过multipass list查看)
ip -a
#退出vm
exit
#安装远程桌面
brew install --cask microsoft-remote-desktop
#点击Add PC，填写vm的ip地址，之后即可使用Ubuntu的图形界面（虽然有点卡）。
```

所以我还是选择用 Docker 搭建环境。

Docker

> Docker 大法好！

Docker 下载

首先下载 Docker Desktop。

1. 可以通过官网下载——戳这里。
2. 也可以通过万能的 Homebrew 下载。

```
brew install --cask docker
```

Ubuntu 镜像拉取

如果没有特殊需求的话，可以直接通过 Docker Desktop 的图形界面拉取 Ubuntu 镜像。

或者使用命令:

```
docker pull ubuntu
```

默认拉取基于 arm64 的最新版本 Ubuntu 镜像（当前是 Ubuntu 22.04 LTS）

但因为我们要拉取的 Ubuntu 镜像不是系统默认的 Ubuntu arm64，所以需要特别指定版本。我们可以使用如下命令

```
docker pull --platform linux/amd64 ubuntu:latest
```

- --platform：Ubuntu 基于的系统和架构，此处为 linux/amd64，当然也可以指定为 linux/arm64。
- ubuntu:latest：所使用的 Ubuntu 系统的版本，此处为 latest 即最新版本，当然也可以指定为 ubuntu:18.04。

创建容器

可以选择打开 Docker Desktop 的图形化界面，在你的 ubuntu 镜像处点击 run 按钮来创建一个新的容器，但这样做毕竟不方便我们设置一些东西。所以还是建议使用命令行来操作。

```
docker container run -it -p 7777:22 -v /Users/xxx/Desktop/CS:/CS --
name=csapp_env  ubuntu:latest /bin/bash
```

- -i：-interactive 交互式操作。
- -t：-tty，分配一个伪终端
- -p：-publish list，主机端口:容器端口
- -v：-volume list，绑定挂载一个卷，即在容器中可以对挂载的（主机中的）文件夹进行操作。使用 -mount 可以将整个主机的文件系统附加到容器。
- --name：指定容器的名称。
- ubuntu:latest：在该镜像中创建容器
- /bin/bash：放在镜像名后的是命令，这里我们希望有个交互式 Shell，因此用的是 /bin/bash，如果你想的话，也可以放 hello_world.c。

Docker 镜像中 CS:APP 环境的配置

可以通过 Docker Desktop 唤起容器并打开终端，也可以使用如下命令：

```
docker exec -it xxx /bin/sh
```

- xxx：容器的 ID，用于指定容器，可以用 `docker ps`或者 Docker Desktop 来查看。
- /bin/sh：指定的终端，也可以是 /bin/bash 或 /bin/zsh 之类的。

apt-get 配置

> Advanced Package Tool，又名 apt-get ，是一款适用于 Unix 和 Linux 系统的应用程序管理器。

在 Ubuntu 中起到 Mac 上 Homebrew 的作用。

```
#更新apt-get的源
apt-get update
#由于docker拉取的Ubuntu镜像中甚至不存在sudo和vim，所以需要重新安装。
apt-get install sudo
sudo apt-get install vim
```

通过 vim 我们可以更改 Ubuntu 的软件源配置，参考清华大学镜像站。

注意选择对应的 Ubuntu 版本，否则会出现奇怪的 bug。（以下显示的是 Ubuntu:22.04LTS 的软件源配置）

```
# 默认注释了源码镜像以提高 apt update 速度，如有需要可自行取消注释
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy main restricted
universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy main restricted
universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-updates main
restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-updates main
restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-backports main
restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-backports main
restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-security main
restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-security main
restricted universe multiverse

# 预发布软件源，不建议启用
# deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-proposed main
restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-proposed main
restricted universe multiverse
```

然后通过 vim/emacs 编辑文件。

```
#备份源文件
cp /etc/apt/sources.list /etc/apt/sources.list.bak
#修改源文件
vim /etc/apt/sources.list
#更新软件源列表和本地已安装的软件，使源文件生效
sudo apt-get update
sudo apt-get upgrade
```

现在应该就能愉快地下载啦！

当然，如果你还嫌不够的话，还可以配置代理。（不过用代理的话容易出 bug，有时候 `unset`或者 `sudo`一下就好有时候不管怎么配都是 `connection error`，所以我最后还是裸奔了）

容器代理设置

- 在 `Docker Desktop->Preference->Resources->Proxies` 或者 `~/.docker/config.json` 中设置代理。在 .json 文件中添加如下设置（7890 是 clash 的默认配置）：

```
{
 "proxies":
 {
   "default":
   {
     "httpProxy": "http://127.0.0.1:7890",
     "httpsProxy": "http://127.0.0.1:7890",
     "noProxy": "localhost,127.0.0.1,.example.com"
   }
 }
}
```

- 在 `Docker Desktop->Preference->Docker Engine` 或者 `~/.docker/daemon.json` 中配置 dns。在 .json 文件中添加如下配置：

```
{
    "dns": ["自建DNS服务的IP", "8.8.8.8"]
}
```

接下来配置 CS:APP 的编译环境。此处参考这个教程。

CS:APP 编译环境配置

```
#安装c/c++编译环境
sudo apt-get install build-essential
#补充gcc的完整环境(gcc-multilib)
sudo apt-get install gcc-multilib
#安装gdb
sudo apt-get install gdb
```

SSH 服务配置

通过 SSH 就可以在万能的 VScode 上跑容器里的代码啦！

```
#安装openssh
sudo apt-get install -y openssh-server
```

```
#配置ssh
vim /etc/ssh/sshd_config
#修改SSH配置文件，增加以下内容
Port 22                         #开启22端口
PermitRootLogin yes             #允许root用户使用ssh登录
RSAAuthentication yes           #启用 RSA 认证
PubkeyAuthentication yes        #启用公钥私钥配对认证方式
AuthorizedKeysFile              .ssh/authorized_keys .ssh/authorized_keys2
#公钥文件路径
#重启ssh服务
service ssh restart

#SSH服务自启动设置
vim /root/startup_run.sh
#写入以下内容(关于zsh，现在当然是没有zsh的，所以可以先看后面的内容配置一下zsh)

#!/bin/zsh
LOGTIME=$(date "+%Y-%m-%d %H:%M:%S")
echo "[$LOGTIME] startup run..." >>/root/startup_run.log
service ssh start >>/root/startup_run.log
#service mysql start >>/root/startup_run.log

#增加文件权限
chmod +x /root/startup_run.sh
#打开启动文件
vim /root/.zshrc
#把脚本命令添加到文件末尾：

# startup run
if [ -f /root/startup_run.sh ]; then
      /root/startup_run.sh
fi

#生效.zshrc(~:root)
source ~/.zshrc
```

VScode 连接容器

1. 安装插件 `remote-ssh`。
2. `control+shift+p`，输入 `open Configuration file` ,填写配置文件。

```
Host CSAPP //服务器名字，可任取
  HostName 127.0.0.1//服务器地址，本地为127.0.0.1
  User root
  Port 7777//创建容器时设置的映射端口
```

3. 打开 VScode 的远程资源管理器的 `SSH TARGETS`，连接 CSAPP。

4. 当然，你也可以不通过 SSH 来连接容器，打开 VScode 远程资源管理器，选择 `Containers`,你可以看到所有你建立的容器，点击连接就好了。

5. 另外，你也可以在 VScode 中安装 Docker 扩展，更加方便地对你的容器和镜像进行操作。

容器/镜像的导入、导出与推送

好不容易配好的环境，当然不能因为手滑而被 Delete 掉，所以肯定要进行一些备份的操作啦~

容器的导入与导出

导出某个容器。

```
docker export xxxx > csapp_env.tar
```

- `xxxx` 为该容器的 ID。
- `csapp_env.tar` 为导出文件的文件名与格式
- 文件默认导出至 `~/`

导入某个容器的快照

```
docker import csapp_env.tar csapp_env:v1
```

- csapp_env.tar 为要导入的快照，注意在快照所在文件路径下操作
- csapp_env 为导入后该镜像的名称
- v1 为镜像的 tag
- 注意，快照导入后变为镜像而非容器

镜像的导入与导出

导出某个镜像

```
docker save ubuntu:latest > ubuntu.tar
```

- ubuntu:latest 为要打包的镜像的名称

- ubuntu.tar 为打包后的归档的名称，默认存储于 `~/`

载入镜像包

```
docker load < ubuntu.tar
```

- ubuntu.tar 为要载入的镜像包的名称
- 注意在镜像包文件所在路径下操作

推送镜像

push 前，将某个镜像改名为 `username/imagename`，否则会报错 `errors: denied: requested access to the resource is denied unauthorized: authentication required`

```
docker tag csapp_env:v1 username/csapp_env:v1 # 其中，username 指你所登入的 docker hub 的用户名。
```

推送镜像至你的 docker hub

- 可以通过 Docker Desktop 的 `Push to Hub`选项推送镜像。
- 也可以通过命令行。

  ```
  docker push username/csapp_env:v1
  ```

之后就可以从 `Docker Desktop->Images->REMOTE REPOSITORIES->username->username/csapp_env:v1` 处拉取镜像。 或者通过命令行拉取。

```
docker pull username/csapp_env:v1
```

常用的 Docker 命令汇总

那么多命令查起来太麻烦了，不如汇总一下～（其实只不过是把菜鸟教程的内容整理了一遍）

显示｜查询各种信息

ps

列出所有在运行的容器信息

```
docker ps
```

具体显示内容如下：

```
CONTAINER ID   IMAGE                     COMMAND                  CREATED
STATUS                  PORTS                  NAMES
540ab7847355   dyego/snake-game          "/bin/sh -c \"./_bin/…"   2 days ago
Up 24 hours (Paused)                           compassionate_newton
dc924cf78ea7   ubuntu:latest             "bash"                   2 days ago
Up 24 hours             0.0.0.0:7778->23/tcp   test
```

images

列出本地镜像

```
docker images
```

具体显示内容如下：

```
REPOSITORY              TAG       IMAGE ID       CREATED       SIZE
ubuntu                  latest    27941809078c   6 weeks ago   77.8MB
```

inspect

获取容器/镜像的元数据

```
docker inspect NAME|ID
```

具体显示内容如下：

```
[
    {
        "Id":
"sha256:27941809078cc9b2802deb2b0bb6feed6c236cde01e487f200e24653533701ee",
        "RepoTags": [
```

```
            "ubuntu:latest"
        ],
        "RepoDigests": [

"ubuntu@sha256:b6b83d3c331794420340093eb706a6f152d9c1fa51b262d9bf34594887c2c
7ac"
        ],
        "Parent": "",
        "Comment": "",
        "Created": "2022-06-06T22:21:26.309275979Z",
        "Container":
"9d10d71fc8cc81806d2fd79443b0cb3d3fac07aee4fd4e2164d8959fd620ba7e",
        ....
    }
]
```

top

查看容器中运行的进程信息

```
#以下的csapp_env有些指代容器名称有些指代容器ID，请自行判断
docker top csapp_env
```

具体显示内容如下：

```
UID                 PID                 PPID                C
STIME               TTY                 TIME                CMD
root                1868                1841                0
Jul24               ?                   00:00:00            /usr/bin/qemu-
x86_64 /bin/zsh /bin/zsh
```

events

从服务器获取实时事件

```
docker events -f "image"="csapp_env" --since="2022-1-1" --until="2022-1-2"
```

logs

查看容器的日志

```
docker logs --since="2022-1-1" --tail=10 csapp_env
```

port

查看容器的端口映射

```
docker port csapp_env
```

history

查看镜像的创建历史

```
docker history ubuntu:latest
```

info

查看docker系统信息

```
docker info
```

具体显示内容如下：

```
Client:
 Context:    default
 Debug Mode: false
 Plugins:
  buildx: Docker Buildx (Docker Inc., v0.8.2)
  compose: Docker Compose (Docker Inc., v2.6.1)
  extension: Manages Docker extensions (Docker Inc., v0.2.7)
  sbom: View the packaged-based Software Bill Of Materials (SBOM) for an
image (Anchore Inc., 0.6.0)
....
```

version

查看docker版本信息

```
docker version
```

具体显示内容如下：

```
Client:
 Cloud integration: v1.0.24
 Version:           20.10.17
 ....
Server: Docker Desktop 4.10.1 (82475)
 Engine:
  Version:          20.10.17
  ....
 containerd:
  Version:          1.6.6
  ....
 runc:
  Version:          1.1.2
  ....
 docker-init:
  Version:          0.19.0
  ....
```

容器管理

run

新建一个容器并运行一个命令

```
docker run [OPTIONS] IMAGE_NAME [COMMAND][ARG...]
```

OPTIONS:

- -a stdin : 指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；
- -d : 后台运行容器，并返回容器 ID；
- -i : 以交互模式运行容器，通常与 -t 同时使用；
- -P : 随机端口映射，容器内部端口随机映射到主机的端口
- -p : 指定端口映射，格式为：主机(宿主)端口:容器端口
- -t : 为容器重新分配一个伪输入终端，通常与 -i 同时使用；
- --name="nginx-lb" : 为容器指定一个名称；
- --dns 8.8.8.8 : 指定容器使用的 DNS 服务器，默认和宿主一致；
- --dns-search DomainName : 指定容器 DNS 搜索域名，默认和宿主一致；
- -h "mars" : 指定容器的 hostname；
- -e username="ritchie" : 设置环境变量；
- --env-file=*FILENAME* : 从指定文件读入环境变量；
- --cpuset="0-2" or --cpuset="0,1,2" : 绑定容器到指定 CPU 运行；
- -m : 设置容器使用内存最大值；

- --net="bridge" : 指定容器的网络连接类型，支持 bridge/host/none/container 四种类型；
- --link=*CONTAINER* : 添加链接到另一个容器；
- --expose=*PORT* : 开放一个端口或一组端口；
- --volume , -v : 绑定一个卷

exec

在运行的容器中执行命令(常用于唤起终端)

```
docker exec -it xxxx /bin/zsh
```

- -it:交互模式运行
- xxxx:容器ID
- /bin/zsh:要执行的命令

start/stop/restart/kill

- docker start :启动一个或多个已经被停止的容器
- docker stop :停止一个运行中的容器
- docker restart :重启容器
- docker kill :杀掉一个运行中的容器

```
docker start csapp_env
docker stop csapp_env
docker restart csapp_env
docker kill csapp_env
```

attach

连接到正在运行中的容器

```
docker attach csapp_env
```

rm

删除一个或多个容器

```
```

```
#强制删除容器c1,c2和它们挂载的数据卷
docker rm -f -v csapp_env_1 csapp_env_2
#删除容器间的连接rlt_name
docker rm -l rlt_name
```

- -f : 强制删除运行中的容器
- -l : 移除容器间的网络连接，而非容器本身
- -v : 删除容器挂载的数据卷。

pause/unpause

- docker pause : 暂停容器中所有的进程
- docker unpause : 恢复容器中所有的进程

```
docker pause csapp_env
docker unpause csapp_env
```

creat

创建一个新的容器但不启动它，用法同run

```
docker create [OPTIONS] IMAGE [COMMAND] [ARG...]
```

wait

阻塞运行直到容器停止，然后打印出它的退出代码

```
docker wait csapp_env
```

export/import

- docker export : 将文件系统作为一个tar归档文件导出到STDOUT
- docker import : 从归档文件中创建镜像

```
#export
docker export csapp_env > csapp_env.tar
#import
docker import csapp_env.tar csapp_env:v1
```

镜像仓库管理（远程）

这些命令都可以被 Docker Desktop 的操作替代。

login/logout

- docker login : 登陆到一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub
- docker logout : 登出一个Docker镜像仓库，如果未指定镜像仓库地址，默认为官方仓库 Docker Hub

```
#登陆至Docker Hub
docker login -u username -p password
docker logout
```

pull/push

- docker pull : 从镜像仓库中拉取或者更新指定镜像
- docker push : 将本地的镜像上传到镜像仓库,要先登陆到镜像仓库

```
#pull
docker pull --platform linux/amd64 ubuntu:latest
#push
docker push username/csapp_env
```

search

从 Docker Hub 查找镜像

```
#在Docker Hub中查找所有收藏数不少于10且镜像名包含ubuntu的镜像
docker search -f stars=10 ubuntu
```

镜像管理（本地）

同样的，这些命令大多也可以由 Docker Desktop 实现，而且更方便。

rmi

删除本地一个或多个镜像。

```
#强制删除image_1:v1,image_2:v2
docker rmi -f image_1:v1 image_2:v2
```

tag

标记本地镜像，将其归入某一仓库

```
#将镜像ubuntu:latest标记为 username/ubuntu:v1 镜像
docker tag ubuntu:latest username/ubuntu:v1
```

build

docker build 命令用于使用 Dockerfile 创建镜像

```
docker build [OPTIONS] PATH | URL | -
#使用当前目录的 Dockerfile 创建镜像，标签为 username/ubuntu:v1。
docker build -t username/ubuntu:v1 .
#使用URL github.com/creack/docker-firefox 的 Dockerfile 创建镜像。
docker build github.com/creack/docker-firefox
#通过 -f 指定Dockerfile 文件的位置：
docker build -f /xxx/xxx/xxx/Dockerfile
```

save/load

- docker save : 将指定镜像保存成 tar 归档文件
- docker load : 导入使用 docker save 命令导出的镜像

```
#具体用法见上文
#save
docker save ubuntu:latest > ubuntu.tar
#load
docker load ubuntu:v1 <ubuntu.tar
```

Shell 配置

理论上来讲，到这一步就可以做 CS:APP 的所有 lab 了，但是这当然是不够的。毕竟用惯了调教好的 shell ， 我已经~~坏掉了~~无法适应这么朴素的界面了，所以顺便配置一下 zsh

和一些常用工具（乐子）。

我个人比较中意的 shell 是 zsh ， 只能说不愧是 shell 中的极品。此处参考这个教程， 这个教程和这个教程。

zsh 下载

在下载 zsh 之前，不妨先看看你到底有哪些 shell：

```
cat etc/shells
```

然后安装 zsh

```
sudo apt-get install zsh
```

并将默认的 shell 改为 zsh。

```
chsh -s /bin/zsh
```

现在只要输入 `zsh`就可以进入 `Z-Shell` 啦！

oh-my-zsh 配置

- 在安装 oh-my-zsh 之前,我们还要先安装一下 wget。

  ```
  sudo apt-get install wget
  #甚至还可能要先安装一下git
  sudo apt-get install git
  ```

- 通过 wget 下载 oh-my-zsh

  ```
  wget https://github.com/robbyrussell/oh-my-
  zsh/raw/master/tools/install.sh -O - | sh
  #gitee加速
  wget https://gitee.com/mirrors/oh-my-zsh/raw/master/tools/install.sh -O
  - | sh
  ```

```
#通过脚本自动安装
bash ./install.sh
```

- 更改 zsh 的主题：

```
#编辑.zshrc文件中的ZSH_THEME项
vim ~/.zshrc
#默认的配置为：
ZSH_THEME="robbyrussell"
#个人比较喜欢的配置为
ZSH_THEME="ys"
#可以通过如下命令查看zsh支持的主题
ls ~/.oh-my-zsh/themes
```

- 除此之外，还有一些必备的插件需要安装。
  - zsh-autosuggestions: 历史命令建议插件。

    ```
    git clone https://gitee.com/phpxxo/zsh-autosuggestions.git
    ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions
    ```

  - zsh-syntax-highlighting: 命令行语法高亮插件。

    ```
    git clone https://gitee.com/Annihilater/zsh-syntax-
    highlighting.git ${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-
    syntax-highlighting
    ```

- 配置 ~/.zshrc

```
# Path to your oh-my-zsh installation.
export ZSH="~/.oh-my-zsh"
#命令行主题
ZSH_THEME="ys"

# 以下内容去掉注释即可生效：
# 启动错误命令自动更正
ENABLE_CORRECTION="true"
# 在命令执行的过程中，使用小红点进行提示
COMPLETION_WAITING_DOTS="true"

# 要使用的插件
plugins=(
        git
        zsh-autosuggestions
        zsh-syntax-highlighting
)
```

```
source $ZSH/oh-my-zsh.sh
source $ZSH_CUSTOM/plugins/zsh-autosuggestions/zsh-autosuggestions.zsh
```

- 使 ~/.zshrc 生效

```
source ~./zshrc
```

好耶！这就得到一个初具人形的 shell 啦！ 但是，你好不容易搭的 shell 怎么能这么贫瘠呢？来康康一些好玩的终端工具吧！

好玩的终端工具

> 把命令中的 `apt-get`换成 `brew`就能在 mac 上用这些包啦！ （其实这些都是照抄我 mac 的配置的说）

可以参考此教程，此教程，此教程以及此教程。

asciinema：终端会话记录器

参考此教程，此教程，技术文档 和 Github仓库。

好耶！用来录教程或者记录踩过的所有坑（会被蠢哭的吧）。

> asciinema lets you easily record terminal sessions and replay them in a terminal as well as in a web browser.

Installing on Ubuntu

```
sudo apt-add-repository ppa:zanchey/asciinema
sudo apt-get update
sudo apt-get install asciinema
```

Usage

- Record terminal session to demo.cast(local)

```
#录制终端命令至demo.cast
asciinema rec demo.cast
```

- Recording finishes when you exit the shell (hit Ctrl+D or type exit)
- Replay recorded asciicast in terminal
    - Space - toggle pause,
    - . - step through a recording a frame at a time (when paused),
    - Ctrl+C - exit.

```
#重放本地文件:
asciinema play /path/to/demo.cast
#重放远程文件
asciinema play https://asciinema.org/a/jMRNwCJnmJ7UpU8Y7HxJmcoU8
```

- Print full output of recorded asciicast to a terminal

```
#打印整个文件的记录结果至终端
asciinema cat /path/to/demo.cast
```

- Upload recorded asciicast to asciinema.org site

```
#上传录制好的文件至asciinema.org，然后就可以在asciinema.org上维护录制记录，便于观看和分享
asciinema upload demo.cast
```

---

autojump：快速文件目录导航

参考 此教程 和 GitHub仓库。

理论上来讲 `autojump` 似乎可以作为 `zsh` 的插件?

> autojump - a faster way to navigate your filesystem.

Installing on Ubuntu

```
sudo apt-get install autojump
source /usr/share/autojump/autojump.sh on startup
echo '. /usr/share/autojump/autojump.sh'>>~/.zshrc #or ~/.bashrc,depends on
your choice
```

Usage

- autojump 只能让你跳到那些你已经用 cd 到过的目录
- autojump 不能跳到名称以破折号 (-) 开头的目录。

```
j -v #查看安装的 autojump 的版本
j -h #查看帮助选项
j foo #跳转到一个包含foo字符串的目录
jo foo #若你安装有Linux桌面环境，可以在终端直接打开包含foo字符串目录的文件管理器
jc foo #跳转到一个包含foo字符串目录的子目录
j --stat #查看每个文件夹的权重和全部文件夹计算得出的总权重的统计数据
j #进入权重最高的目录

#改变权重值:
j -i [权重] #增加
j -d [权重] #减少

j --purge #去除不存在的路径
jco c #在文件管理器中打开一个子目录
```

---

axel : 轻量级多线程下载工具

参考 此教程 和 GitHub仓库。

加速！加速！给我加速！

> Axel tries to accelerate the download process by using multiple connections per file, and can also balance the load between different servers.

Installing on Ubuntu

```
sudo apt-get update
sudo apt-get install axel
```

Usage

```
# 开启10个线程下载
axel -n 10 https://github.com/axel-download-
accelerator/axel/archive/refs/heads/master.zip
```

---

bat/fx/hexyl : 各种文件查看器

但有什么是 Vim 不能替代的吗?

bat : cat++

参考 此教程 和 Github仓库。

> A cat clone with wings.

Installing on Ubuntu

```
sudo apt-get install bat
#这里要注意一下，由于名称的冲突，在 Ubuntu 下无法直接通过敲 bat 这三个字母运行 bat 命
令，而是需要敲 batcat 。为了方便，可以设置别名
alias bat="batcat"
```

Usage

bat 主要功能:

- 语法高亮——支持根据编程语言以及 markdown 语法将文本关键词高亮，可阅读性大大增强;
- 自动翻页——如果文本内容太长，超过一页的话，它会自动将内容通过管道传到 less 命令，实现翻页的效果;
- 集成Git——bat 命令跟 git 完美结合，将文本的修改部分在左侧展示，一眼就可以看出文件修改了啥。
- 显示不可打印字符——有些字符无法通过 cat 在屏幕上显示

```
#使用方法与cat基本相同
#bat 对大部分编程语言和标记语言提供语法高亮，可读性更强。
#查看所有支持的语言:
bat --list-languages
#显示不可打印字符
bat -A xxx.txt
#主题定制
```

```
#bat 命令支持多种风格的语法高亮主题，可以满足大部分需求。
#查看 bat 支持的所有主题
bat --list-themes
#如果想指定主题为 GitHub ，可以使用以下命令
bat --theme=GitHub 文件名
#但这样做是临时的，关闭终端工具或者系统重启，主题就会恢复默认。可以通过设置 BAT_THEME 环境变量来选定主题。
#把 export BAT_THEME="GitHub" 添加到 shell 的启动脚本来取得永久效果。除此之外，还可以自定义输出样式，甚至可以自己添加新的语言和语法，输出让自己满意的高亮样式。
```

jq : json 文件查看与编辑

参考 此教程 和 官网。 jq 是 JSON 处理神器!

> jq is like sed for JSON data - you can use it to slice and filter and map and transform structured data with the same ease that sed, awk, grep and friends let you play with text.

Installing on Ubuntu

```
sudo apt-get install jq
```

Usage

```
# 格式化查看数据
cat /data/test.json | jq
# 格式化重输到新文件中
cat /data/test.json | jq .
# 格式化文件，并重命名新文件
cat /data/test.json | jq . > test.json
jq '.' json.txt — 格式化
jq '.[0].address.city' json.txt #获取指定字段的值(可嵌套)
jq '.[0].name,.[0].url' json.txt #使用逗号分隔，取多个字段的值；返回值以换行分隔
jq '.[0].arrayBrowser[0]' json.txt #提取数组的单个元素
jq '.[0].arrayBrowser[0,1]' json.txt #提取数组的多个元素
jq '.[0].arrayBrowser[]' json.txt #提取数组的全部元素，返回值以换行分隔
jq '[.data.results[]|{topicId,viewCount}]' hjsource.json #提取数组的指定属性
jq '[.data.results[]|{id:topicId,count:viewCount}]' hjsource.json #提取数组的指定属性并重命名
```

hexyl : 16进制文件查看器

参考 此教程 和 GitHub仓库。 傻瓜式地查看16进制文件或以16进制查看文件!

> hexyl is a simple hex viewer for the terminal. It uses a colored output to distinguish different categories of bytes (NULL bytes, printable ASCII characters, ASCII whitespace characters, other ASCII characters and non-ASCII).

Installing and Usage

```
#Installing
sudo apt-get install hexyl
#Usage
#支持png, txt等等
hexyl [filename]
#例子
hexyl t.txt
```

```
┌────────┬─────────────────────────┬─────────────────────────┬────────┬────┐
└────┘
│00000000│ 48 65 6c 6c 6f 20 57 6f ┊ 72 6c 64 21 0a 49 20 6c │Hello
Wo┊rld!_I l│
│00000010│ 6f 76 65 20 4c 69 6e 75 ┊ 78 21                   │ove Linu┊x!
│
└────────┴─────────────────────────┴─────────────────────────┴────────┴────┘
└──┘
```

---

boxes：命令行字符形状工具

参考 GitHub仓库 和 官网。 帅耶！虽然没什么实际用处，但在脚本或者什么其他地方用`boxes`输出信息总是让人有种成为大佬的错觉。

> Boxes is a command line filter program which draws ASCII art boxes around your input text.

Installing on Ubuntu

```
sudo apt-get install -y boxes
```

Usage

```
#Draws a standard box of the first valid design found in your config file.
boxes
/*************************/
/* Different all twisty a */
/* of in maze are you,    */
/* passages little.       */
```

```
/*************************/
#As a text filter, boxes can be combined easily with other tools, for
example figlet and lolcat
figlet "boxes . . . !" | lolcat -f | boxes -d unicornthink
#The -d design_name option selects another design from your config file.
boxes -d parchment
 _________________________
/\                         \
\_|Different all twisty a|
  |of in maze are you,    |
  |passages little.       |
  |   ____________________|_
   \_/_____________________/
#The -a format option can be used to position the input text inside a box
which is larger than needed for our text. hcvc stands for "horizontally
centered, vertically centered".
boxes -d diamonds -a hcvc
       /\          /\          /\
    /\//\\/\    /\//\\/\    /\//\\/\
 /\//\\\///\\/\//\\\///\\/\//\\\///\\/\
//\\\//\\\///\\\//\\\///\\\//\\\//\\\//\\
\\//\/                             \/\\//
 \/                                   \/
 /\      Different all twisty a       /\
//\\     of in maze are you,         //\\
\\//     passages little.            \\//
 \/                                   \/
 /\                                   /\
//\\/\                             /\//\\
\\///\\/\//\\\///\\/\//\\\///\\/\//\\\///
 \/\\//\\\/\/\\\///\\\/\/\\\///\\\/\/
    \/\\//\/    \/\\//\/    \/\\//\/
       \/          \/          \/
#This uses the third possible argument to the -a option, j. jr stands for
"justification right".
boxes -d simple -a jr
************************
*Different all twisty a*
*   of in maze are you,*
*      passages little.*
************************
#other examples
boxes -d dog -a c #Quick alignment/positioning
boxes -d html -s 40
boxes -d peek -a c -s 40x11 #Box size specification
boxes -d shell -p a1l2 # Text Padding
```

---

curl : 利用 URL 语法在命令行下工作的文件传输工具

参考 此教程 , 技术文档 和 GitHub仓库。 (虽然有很多用途但本人以前是把它当 `ping` 用的 orz)

> Curl is a command-line tool for transferring data specified with URL syntax.

Installing on Ubuntu

```
sudo apt-get update
sudo apt-get install curl
```

Usage

```
#查看网页源码
curl www.sina.com
#如果要把这个网页保存下来，可以使用`-o`参数，这就相当于使用wget命令了。
curl -o [filename] www.sina.com
#自动跳转
curl -L www.sina.com
#显示头信息，连同网页代码一起。
curl -i www.sina.com
#显示一次http通信的整个过程，包括端口连接和http request头信息。
curl -v www.sina.com
#如果你觉得上面的信息还不够，那么下面的命令可以查看更详细的通信过程。
curl --trace output.txt www.sina.com #运行后，请打开output.txt文件查看
#发送表单信息
curl example.com/form.cgi?data=xxx
#文件上传
curl --form upload=@localfilename --form press=OK [URL]
#HTTP动词:curl默认的HTTP动词是GET，使用`-X`参数可以支持其他动词。
curl -X POST www.example.com
#Referer字段:有时你需要在http request头信息中，提供一个referer字段，表示你是从哪里跳转过来的。
curl --referer http://www.example.com http://www.example.com
#User Agent字段:这个字段是用来表示客户端的设备信息。服务器有时会根据这个字段，针对不同设备，返回不同格式的网页，比如手机版和桌面版。
curl --user-agent "[User Agent]" [URL]
#cookie:使用`--cookie`参数，可以让curl发送cookie。
curl --cookie "name=xxx" www.example.com
#增加头信息:有时需要在http request之中，自行增加一个头信息。`--header`参数就可以起到这个作用。
curl --header "Content-Type:application/json" http://example.com
#HTTP认证:有些网域需要HTTP认证，这时curl需要用到`--user`参数。
curl --user name:password example.com
```

---

cloc：源码行数统计工具

参考 GitHub仓库。 统计你到底写了多少代码（掉了多少头发），学期末用 `cloc` 总会让人产生满满的成就感～

> cloc counts blank lines, comment lines, and physical lines of source code in many programming languages.

Installing on Ubuntu

```
sudo apt-get update
sudo apt-get install cloc
```

Usage

```
# a file
cloc hello.c
# a direction
cloc gcc-5.2.0/gcc/c
# an archieve
cloc master.zip
# a git repository,using a specific commit
git clone http://git.tiker.net/trees/pudb.git
cd pudb
cloc 6be804e07a5db
# each subdirectory of a particular directory
for d in ./*/ ; do (cd "$d" && echo "$d" && cloc --vcs git); done
./Project0/
# Output Example:
> cloc Desktop/CS
    2700 files
   20872 text files.
   12246 unique files.
   76062 files ignored.

github.com/AlDanial/cloc v 1.94  T=26.98 s (453.8 files/s, 186478.5 lines/s)
-------------------------------------------------------------------------------
-----------
Language                             files          blank        comment
code
-------------------------------------------------------------------------------
-----------
C++                                    796          10563          15401
1297219
C/C++ Header                          2488          87381         160680
820309
Text                                   218          40614              0
540851
Perl                                  1746         136976         158643
532733
Ruby                                  1716          44502          69209
227586
Python                                 605          39580          66741
182840
....
TypeScript                               1              0              0
```

```
3
-------------------------------------------------------------------------------
-----------
SUM:                                  12246         412921         537798
4081151
-------------------------------------------------------------------------------
-----------
```

---

cowsay：萌萌的 ASCII 图片生成器

参考 官网，维基百科 和 此教程。 没什么别的用处，就是为了乐子。

- (或者炫酷的终端)
- (或者大佬范的脚本效果)

> cowsay is a configurable talking cow.

Installing on Ubuntu

```
sudo apt-get install cowsay
```

Usage

```
# 让牛牛说话吧!
cowsay "hello world"
 _____________
< hello world >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
# 查看别的动物参数
cowsay -l

Cow files in /opt/homebrew/Cellar/cowsay/3.04_1/share/cows:
beavis.zen blowfish bong bud-frogs bunny cheese cower daemon default dragon
dragon-and-cow elephant elephant-in-snake eyes flaming-sheep ghostbusters
head-in hellokitty kiss kitty koala kosh luke-koala meow milk moofasa moose
mutilated ren satanic sheep skeleton small stegosaurus stimpy supermilker
surgery three-eyes turkey turtle tux udder vader vader-koala www

# -f <animal_name> : 让别的动物开口说话!
cowsay -f bud-frogs "反思! 反思! "
 ____________________
```

```
< 反思！反思！ >
 --------------------
     \
      \
          oO)-.                       .-(Oo
         /__  _\                     /_  __\
         \  \(  |     ()~()         |  )/  /
          \__|\ |    (-___-)        | /|__/
          '    --     ==`- ==       '--
```

```
# 使用fortune(后面会提到)让牛牛变成哲学家(随机输出语录)
fortune | cowsay
```

```
 _____________________________________
/ Nothing succeeds like success. \
|                                   |
\ -- Alexandre Dumas                /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
#当然也可以让哲学牛牛变成彩色(使用lolcat,后面也会提到)
fortune | cowsay | lolcat
```

---

cmatrix：代码雨！黑客风炫酷屏保软件

参考 此教程 和 GitHub仓库。

黑客帝国欸！虽然只是屏保，但围观你电脑的人不知道啊！

> CMatrix is based on the screensaver from The Matrix website. It shows text flying in and out in a terminal like as seen in "The Matrix" movie. It can scroll lines all at the same rate or asynchronously and at a user-defined speed. CMAtrix is inspired from 'The Matrix' movie. If you haven’t seen this movie and you are a fan of computers or sci-fi in general, go see this movie!!!

Installing on Ubuntu

```
sudo apt-get install cmatrix
```

Usage

```
cmatrix

d         P C     )       8   q 4 i       V         !   N     d   <   Q H K
:
J         ^ ?     z       F   R O h       :         +   ,     a   $     ( I
\
h         T )     /       >   o G         e       ? 9   R     J         I h
o
B         D H     N     P 5   x x         C       D 4   '     f         d V
X
A         ,       E     ] y   ] 9         $       u     & 9   4         b
0
M         o       @     . 2   $ w         )     G H     5 I   D
T
t   B         9   +     % * Z ? 5         Z     E z     + w   c       x
U
"   ?         '   E     j P g J I         <   $ K u     L '   U       M
O
;   N /     m h   G   R Y O F Q >         *   S B w .     `   s       H
J
G ] j q     S S   Z $ - q J j 2 w         )   B G V )     L   W       '
k m
' Q v l     S E   r w j W / D # ,   0     a   t 4 < =     .   #       )
;
] Y t .     _ - a h . : M i W B c   [   R R _ * m b :     5   o       1
W
  Y ; <     0 L < L I F f 2 w u     r x q ! H . S G C     +   ;     g C
'
  B 6 l 4   O k 6 Q O . l G "       k = C   ?   _ P q c 2 +   E     O i ,
r
  = t ? Y   L   ( R ^   "   (       I c M   V   k e ) X D u         v y E
c
  Y z 9 F   S   s : ^   q   7       P , ?   g   # w [ O w y         S ( %
p
  r O   .   &   O <     l   Q       & r D   U   G \ - " S f %       ? : 9
D
  O S   $       =       s   j       V >     q   +   6 * O 8 F       ' L G
W
  U             t       V   ' i   * K 1     J   ^   - < Y < C       W h ` q
%
  T             Z       t   B M   u j _     M   H   f   t   ]       = O m H
3
  T     <       w       _   . C   f ( &     6   _   c   I   f   R   q Z ^ i
*
  '     B       M     i N   1 7   g h d     ]   D   *   W   -   >   O P , P
d
  G   " ]       $   ] h M   d N   Y + N     c   #   +   x   y U a   + * L r
\
      1 :         & V 2   M     P F ?     B i y   y   t   2 L Z   z < [ )
+
```

参数说明：

- -a: 异步滚动
- -b: 粗体字打开

- -B: 所有粗体字符（覆盖-B）
- -c: 使用原始矩阵中的日语字符。需要适当的字体
- -f: 强制启用linux$TERM类型
- -l: Linux模式（使用矩阵控制台字体）
- -L: 锁定模式（可从另一终端关闭）
- -o: 使用旧式滚动
- -h: 打印使用和退出
- -n: 无粗体字符（覆盖-b和-B，默认值）
- -s: “屏幕保护程序”模式，在第一次按键时退出
- -x: x窗口模式，如果您的xterm正在使用mtx.pcf公司
- -V: 打印版本信息并退出
- -u: delay（0-10，默认值4）：屏幕更新延迟
- -C: color_name：将此颜色用于矩阵（默认为绿色）
- -r: 彩虹模式
- -m: lambda模式

运行时按键调整：

- 1-9 :屏幕更新延迟时间，单位秒
- a :异步滚动
- b :粗体字打开
- B :所有字体粗体
- n :关闭粗体
- ! :字体颜色切换为红色
- @ :字体颜色切换为绿色
- # :字体颜色切换为黄色
- $:字体颜色切换为蓝色
- % :字体颜色切换为洋红色
- ^ :字体颜色切换为青色
- & :字体颜色切换为白色
- q :退出屏保

---

ddate：精心调制的混搭日历

因为找不到官方的文档所以就随便口胡一下算了。 不过日历的效果还是挺不错的（起码比千篇一律的xxxx/xx/xx有意思一点)

Installing and Usage

```
#Installing
sudo apt-get install ddate
#Usage
ddate
#Output
Today is Pungenday, the 62nd day of Confusion in the YOLD 3188
```

---

diff-so-fancy : diff++

参考 GitHub仓库。 使得 diff 操作会产生类似 GitHub 的效果。 (不知道为什么在 Ubuntu 里一直配置不好，不过在 Mac 上还是能用的)

> diff-so-fancy strives to make your diffs human readable instead of machine readable. This helps improve code quality and helps you spot defects faster.

Installing on MacOS

```
brew install diff-so-fancy
#将之设置为git diff的默认项
git config --global core.pager "diff-so-fancy | less --tabs=4 -RFX"
git config --global interactive.diffFilter "diff-so-fancy --patch"
```

Usage

```
#设置颜色
git config --global color.ui true

git config --global color.diff-highlight.oldNormal    "red bold"
git config --global color.diff-highlight.oldHighlight "red bold 52"
git config --global color.diff-highlight.newNormal    "green bold"
git config --global color.diff-highlight.newHighlight "green bold 22"

git config --global color.diff.meta       "11"
git config --global color.diff.frag       "magenta bold"
git config --global color.diff.func       "146 bold"
git config --global color.diff.commit     "yellow bold"
git config --global color.diff.old        "red bold"
git config --global color.diff.new        "green bold"
git config --global color.diff.whitespace "red reverse"

#标记空行是否应该着色（默认为真）
git config --bool --global diff-so-fancy.markEmptyLines false
#将git的标头块更易阅读（默认为真）
git config --bool --global diff-so-fancy.changeHunkIndicators false
```

```
#是否带前导符号+/-（默认为真）
git config --bool --global diff-so-fancy.stripLeadingSymbols false
#分隔符使用Unicode或Ascii（默认为真）
git config --bool --global diff-so-fancy.useUnicodeRuler false
#设置文件头的宽度（默认为真）
git config --global diff-so-fancy.rulerWidth 47    # git log's commit header
width

#使用
#与diff连接使用(-u)
diff -u file_a file_b | diff-so-fancy
#git diff
git diff  file_a file_b
──────────────────────────────────────────────────────────────────────────────
────
renamed: t1.txt to t2.txt
──────────────────────────────────────────────────────────────────────────────
────
@ t2.txt:4 @
123
Abc
123
\ No newline at end of file




Adsasdafsas
\ No newline at end of file
```

---

duf：磁盘信息查看工具

此处参考 此教程 和 GitHub仓库。 du 和 df （什么矮人要塞） 的加强版，由于容器是纯命令行界面，所以没法可视化查看磁盘使用情况，这时候 duf 就很有用了。 所以 `Can can need` !

> Disk Usage/Free Utility (Linux, BSD, macOS & Windows)

Installing on Ubuntu

```
#首先，从 GitHub 上下载 duf 命令的安装包：
wget
https://github.com/muesli/duf/releases/download/v0.8.1/duf_0.8.1_linux_amd64
.deb
#然后，再使用 dpkg 命令安装：
dpkg -i duf_0.8.1_linux_amd64.deb
```

Usage

```
#如果你安装了oh-my-zsh，因为你可能已经拥有了名为duf的简称命令（alias），所以请清除它，以免混淆
unalias duf
#直接使用duf命令，让你的磁盘一览无余
#输出所有本地设备、已挂载的任何云存储设备以及任何其他特殊设备（包括临时存储位置等）的详细信息
duf
```

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ 3 local devices                                                              │
├─────────────┬──────┬──────┬───────┬──────┬──────┬──────────────────────────┤
│ MOUNTED ON  │ SIZE │ USED │ AVAIL │ USE% │ TYPE │ FILESYSTEM               │
├─────────────┼──────┼──────┼───────┼──────┼──────┼──────────────────────────┤
│ /etc/hostnam│ 58.4G│ 3.8G │ 51.6G │  6.5%│ ext4 │ /dev/vda1                │
│ e           │      │      │       │      │      │                          │
...
└─────────────┴──────┴──────┴───────┴──────┴──────┴──────────────────────────┘
```

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ 1 fuse device                                                                │
├─────────────┬──────┬──────┬───────┬──────┬──────┬──────────────────────────┤
│ MOUNTED ON  │ SIZE │ USED │ AVAIL │ USE% │ TYPE │ FILESYSTEM               │
├─────────────┼──────┼──────┼───────┼──────┼──────┼──────────────────────────┤
│ /CS         │ 57.1T│ 44.8T│ 12.3T │ 78.5%│ fuse.g│ grpcfuse                │
│             │      │      │       │      │ rpcfus│                         │
│             │      │      │       │      │ e     │                         │
└─────────────┴──────┴──────┴───────┴──────┴──────┴──────────────────────────┘
```

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ 6 special devices                                                            │
├─────────────┬──────┬──────┬───────┬──────┬──────┬──────────────────────────┤
│ MOUNTED ON  │ SIZE │ USED │ AVAIL │ USE% │ TYPE │ FILESYSTEM               │
├─────────────┼──────┼──────┼───────┼──────┼──────┼──────────────────────────┤
│ /dev        │ 64.0M│   0B │ 64.0M │      │ tmpfs│ tmpfs                    │
...
└─────────────┴──────┴──────┴───────┴──────┴──────┴──────────────────────────┘
```

```
#指定特殊的设备，文件，路径或装载点
duf /home /some/file
#列出所有信息(包括伪、重复或不可访问的文件系统)
duf --all
#显示或隐藏特定表格的内容（比如只显示本地设备）
duf --only local,network,fuse,special,loops,binds
duf --hide local,network,fuse,special,loops,binds
#显示或隐藏特定文件系统的内容
duf --only-fs tmpfs,vfat
duf --hide-fs tmpfs,vfat
#显示或隐藏特定文件/路径/装载点的内容
duf --only-mp /,/home,/dev
duf --hide-mp /,/home,/dev
#且若添加引号，则支持通配符
duf --only-mp '/sys/*,/dev/*'
#按占用空间大小排序后输出
duf --sort size
#显示或隐藏特定的列
duf --output mountpoint,size,usage
#除了查看块使用情况，我们还可以通过 --inodes 选项查看 inodes 用量
duf --inodes
```

```
#可以指定duf显示的样式
duf --theme light
#duf可以用各种颜色突出显示某些内容，你可以设置你的阈值
duf --avail-threshold="10G,1G"
duf --usage-threshold="0.5,0.9"
#可以以json的格式显示信息
duf --json
```

---

emacs：最好的编辑器（或许之一）

可以参考此教程，具体内容太多力我先摆为敬。

Installing on Ubuntu

```
sudo apt-get install emacs
```

Usage

> Please Searching-The-Fucking-Web

---

exa：ls的现代替代品

参考GitHub仓库和官网。 比 ls 更加炫酷一些，比如支持彩色显示目录内容等等。但是 `ls` 用惯了，这边只是顺便提一嘴。

> exa is a modern replacement for the venerable file-listing command-line program ls that ships with Unix and Linux operating systems, giving it more features and better defaults. It uses colours to distinguish file types and metadata. It knows about symlinks, extended attributes, and Git. And it's small, fast, and just one single binary.

Installing on Ubuntu

```
sudo apt-get install exa
```

Usage

```
#主要操作其实与ls类似，而且大多数时候都可以用ls解决需求，所以只枚举有特色的两种用法（搬运阮一峰大佬的博客）
#输出目录中文件的git状态
exa --long --git
#以树状结构输出目录（与tree类似）
exa --tree --level=2
```

---

fasd：快速跳转工具

参考 此教程 和 GitHub仓库。 用途与 `autojump` 和 `zoxide` 都有些重合，但似乎又有一些不太一样?

> Fasd (pronounced similar to "fast") is a command-line productivity booster. Fasd offers quick access to files and directories for POSIX shells. It is inspired by tools like autojump, z and v. Fasd keeps track of files and directories you have accessed, so that you can quickly reference them in the command line.

Installing on Ubuntu

```
sudo add-apt-repository ppa:aacebedo/fasd
#如果报错add-apt-repository不存在的话，尝试一下以下命令：
sudo apt-get install software-properties-common
#然后更新列表并安装
sudo apt-get update
#当然也有可能，你可以直接安装fasd
sudo apt-get install fasd
```

Settings

```
#要想使fasd工作，首先请将以下命令添加至.zshrc文件
eval "$(fasd --init auto)"
#如果安装了oh-my-zsh，请在.zshrc文件中将fasd添加至oh-my-zsh的插件
plugins=(... fasd)
#可以选择将以下内容添加进.zshrc的配置
fasd_cache="$HOME/.fasd-init-zsh"
if [ "$(command -v fasd)" -nt "$fasd_cache" -o ! -s "$fasd_cache" ]; then
  fasd --init posix-alias zsh-hook zsh-ccomp zsh-ccomp-install >|
"$fasd_cache"
fi
source "$fasd_cache"
unset fasd_cache
#取消以下alias，否则会与fzf产生冲突
unalias z
```

```
unalias zz
#如果您想对进入shell环境的内容有更多的控制，可以将定制的参数集传递给fasd-init
#参数如下
zsh-hook            # define _fasd_preexec and add it to zsh preexec array
zsh-ccomp           # zsh command mode completion definitions
zsh-ccomp-install   # setup command mode completion for zsh
zsh-wcomp           # zsh word mode completion definitions
zsh-wcomp-install   # setup word mode completion for zsh
bash-hook           # add hook code to bash $PROMPT_COMMAND
bash-ccomp          # bash command mode completion definitions
bash-ccomp-install  # setup command mode completion for bash
posix-alias         # define aliases that applies to all posix shells
posix-hook          # setup $PS1 hook for shells that's posix compatible
tcsh-alias          # define aliases for tcsh
tcsh-hook           # setup tcsh precmd alias
#示例如下
eval "$(fasd --init posix-alias zsh-hook)"
#最后，使配置生效
source ~/.zshrc
```

Usage

```
fasd [options] [query ...]

[f|a|s|d|z] [options] [query ...]
  options:
    -s          list paths with scores
    -l          list paths without scores
    -i          interactive mode
    -e <cmd>    set command to execute on the result file
    -b <name>   only use <name> backend
    -B <name>   add additional backend <name>
    -a          match files and directories
    -d          match directories only
    -f          match files only
    -r          match by rank only
    -t          match by recent access only
    -R          reverse listing order
    -h          show a brief help message
    -[0-9]      select the nth entry

fasd [-A|-D] [paths ...]
    -A    add paths
    -D    delete paths

#Examples
f foo           # list frecent files matching foo
a foo bar       # list frecent files and directories matching foo and bar
f js$           # list frecent files that ends in js
f -e vim foo    # run vim on the most frecent file matching foo
mplayer `f bar` # run mplayer on the most frecent file matching bar
z foo           # cd into the most frecent directory matching foo
open `sf pdf`   # interactively select a file matching pdf and launch `open`
```

fd : find++

参考 此教程 和 GitHub仓库。 比 `find` 更快，更符合直觉。

> fd is a program to find entries in your filesystem. It is a simple, fast and user-friendly alternative to find. While it does not aim to support all of find's powerful functionality, it provides sensible (opinionated) defaults for a majority of use cases.

Installing on Ubuntu

```
sudo apt-get install fd-find
#命令名为fdfind，防止指令冲突，但是如果你想的话也可以alias一下
alias fdfind="fd"
```

Usage

```
#若不带命令行选项，则输出当前文件夹下的所有文件（包含子文件夹里，采用递归搜索），相当于后面会介绍的tree或前面讲到的exa --tree
fdfind
#查看特定类型的文件
fdfind -e cpp
#查找特定文件
fdfind test.c
#在特定路径中查找文件（在/etc中查找文件名包含foo的文件）
fdfind foo /etc
#忽略或不忽略.gitignore文件所指定的文件
fdfind -I num_cpu
fdfind --no-ignore num_cpu
#默认情况下,fd不搜索隐藏目录,不在搜索结果中显示隐藏文件. 若要禁用此行为,我们可以使用-H（或）--hidden选项:
fdfind -H t.c
#如果我们想在所有搜索结果上运行命令,我们可以连接xargs。这里,-0选项告诉fd用空字符（而不是换行符）分隔搜索结果. 以同样的方式,xargs的-0选项同样告诉它以这种方式读取输入.
fd -0 -e rs | xargs -0 wc -l
#fd还可与fzf的集成：支持模糊搜索

#具体的命令行选项如下:
USAGE:
    fd [FLAGS/OPTIONS] [<pattern>] [<path>...]

FLAGS:
    -H, --hidden            搜索隐藏的文件和目录
    -I, --no-ignore         不要忽略 .(git | fd)ignore 文件匹配
        --no-ignore-vcs     不要忽略.gitignore文件的匹配
    -s, --case-sensitive    区分大小写的搜索（默认值：智能案例）
    -i, --ignore-case       不区分大小写的搜索（默认值：智能案例）
    -F, --fixed-strings     将模式视为文字字符串
```

```
    -a, --absolute-path      显示绝对路径而不是相对路径
    -L, --follow             遵循符号链接
    -p, --full-path          搜索完整路径（默认值：仅限 file-/dirname）
    -0, --print0             用null字符分隔结果
    -h, --help               打印帮助信息
    -V, --version            打印版本信息

OPTIONS:
    -d, --max-depth <depth>          设置最大搜索深度（默认值：无）
    -t, --type <filetype>...         按类型过滤：文件（f），目录（d），符号链接
 (l)，
                                     可执行（x），空（e）
    -e, --extension <ext>...         按文件扩展名过滤
    -x, --exec <cmd>                 为每个搜索结果执行命令
    -E, --exclude <pattern>...       排除与给定glob模式匹配的条目
        --ignore-file <path>...      以.gitignore格式添加自定义忽略文件
    -c, --color <when>               何时使用颜色：never，*auto*，always
    -j, --threads <num>              设置用于搜索和执行的线程数
    -S, --size <size>...             根据文件大小限制结果。

ARGS:
    <pattern>    the search pattern, a regular expression (optional)
    <path>...    the root directory for the filesystem search (optional)
```

---

figlet/toilet：终端艺术字

> 由于篇幅限制，此处仅介绍 figlet（开摆！）

参考 此教程 和 官网。 不觉得很酷吗？作为一名理工男我觉得这太酷了，很符合我对未来生活的想象。科技并带着趣味。

> FIGlet is a program that creates large characters out of ordinary screen characters.

Installing on Ubuntu

```
sudo apt-get install figlet
```

Usage

```
#简单使用
figlet "Hello World"
# _   _      _ _        __        __         _     _
#| | | | ___| | | ___   \ \      / /__  _ __| | __| |
#| |_| |/ _ \ | |/ _ \   \ \ /\ / / _ \| '__| |/ _` |
#|  _  |  __/ | | (_) |   \ V  V / (_) | |  | | (_| |
```

```
#|_| |_|\___|_|_|\___/     \_/\_/ \___/|_|  |_|\__,_|

#查看可用字体
showfigfonts
#指定字体进行显示
figlet -f block "Hello World"
_|    _|            _|  _|
_|    _|    _|_|    _|  _|    _|_|
_|_|_|_|  _|_|_|_|  _|  _|  _|    _|
_|    _|  _|        _|  _|  _|    _|
_|    _|    _|_|_|  _|  _|    _|_|



_|          _|                      _|        _|
_|          _|    _|_|    _|  _|_|  _|    _|_|_|_|
_|    _|    _|  _|    _|  _|_|      _|  _|    _|
  _|  _|  _|    _|    _|  _|        _|  _|    _|
    _|  _|        _|_|    _|        _|    _|_|_|

#当然，更有趣的是与其他命令连接使用
#譬如与lolcat连接得到彩色字
figlet "Hello World" | lolcat
#又譬如与fortune连接得到哲人语录的艺术字版
fortune | figlet
#或者与watch和date连用，获得艺术字时钟
watch -n1 "date '+%D%n%T'|figlet -k"

Every 1.0s: date '+%D%n%T'|fig...  shouchehendeMBP.lan: Wed Aug  3 15:30:58
2022


  ___     ___      __ ___  _____     __ ____   ____
 / _ \   ( _ )    / // _ \|___ /    / /|___ \ |___ \
| | | | / _ \   / /| | | | |_ \   / /   __) |  __) |
| |_| || (_) |/ / | |_| |___) |/ /   / __/  / __/
 \___/  \___//_/   \___/|____//_/   |_____||_____|


 _  ____      _____   ___      ____   ___
/ ||___ \ _  |___ /  / _ \    | ___| / _ \
| |  __) (_)   |_ \ | | | |(_)|___ \| (_) |
| | / __/ |_  ___) || |_| | _  ___) |\__, |
|_||_____/(_)|____/  \___/ (_)|____/   /_
```

---

fortune：随机名言生成器

参考 此教程，官方技术文档 和 GitHub仓库。 中学生狂喜！终于不用45度明媚忧伤地在淡蓝色笔记本上记名人名言了！

> Fortune is a simple Unix program that displays a random message from a database of quotations.

Installing on Ubuntu

```
#安装fortune
sudo apt-get install fortune
#添加fortune的中文库
# Debian/Ubuntu
git clone git@github.com:ruanyf/fortunes.git
sudo mv fortunes/data/* /usr/share/games/fortunes/
# Mac
git clone git@github.com:ruanyf/fortunes.git
strfile fortunes/data/fortunes
strfile fortunes/data/chinese
strfile fortunes/data/tang300
strfile fortunes/data/song100
strfile fortunes/data/diet
mv fortunes/data/* /usr/local/share/games/fortunes/ #如果是用brew下载的话，后一项的内容可以换做/opt/homebrew/Cellar/fortune/9708/share/games/fortunes之类的试试
```

Usage

```
#(主要参考阮一峰的博客)
#使用
fortune
#设置长格言的最短长度（默认为160），比这更短的就是短格言
fortune -n LENTH
#打印长格言
fortune -l
#打印短格言
fortune -s
#与其他命令连接（生成彩色艺术字）
fortune | lolcat | figlet
#查看每个格言库所占的比重
fortune -f
100.00% /opt/homebrew/Cellar/fortune/9708/share/games/fortunes
     0.54% song100
     5.41% computers
     ...
     0.10% translate-me
#在 ~/.bashrc 或 ~/.zshrc 文件（根据你使用的 shell 而定）的结尾，加上下面几行，那么每次启动 shell 窗口，就会自动跳出一句格言
echo
echo "=============== Quote Of The Day ==============="
echo
fortune
echo
echo "================================================="
echo

#如果你不满足于当前的格言库的话，可以尝试自己制作格言包。
#格言包就是一个文本文件，可以放入任何内容。假如你想用它背单词，也没有问题。下面就是如何制作这一类的 fortune 数据文件。
#（1）所有条目都写入一个文本文件，文件名任意。
#（2）条目之间用单独一行的百分号（%）分隔，就像下面这样。
```

```
路漫漫其修远兮，吾将上下而求索。
----屈原《离骚》
%
富贵不能淫，贫贱不能移，威武不能屈。
----《孟子》
%
长风破浪会有时，直挂云帆济沧海。
----《行路难·其一》
# (3) 生成索引文件。
strfile <fortuneFile> <fortuneFile.dat>
#上面命令中，尖括号的 fortune 文件名，替换成你的文件名。
# (4) fortune 数据文件和它的索引文件，都拷贝到目录 /usr/share/games/fortunes/(Mac
上是/opt/homebrew/Cellar/fortune/9708/share/games/fortunes) 。
```

---

fzf：命令行模糊查找神器

参考 此教程，此教程 和 GitHub仓库。 看起来好方便的样子！ （但感觉学起来好累嘤嘤嘤，所以Ubuntu上的配置就不学了嘤嘤嘤）

> fzf is a general-purpose command-line fuzzy finder.

下文中所有关于 fzf 配置的内容请写入 `~/.fzf.zsh`，之后在 `~/.zshrc` 文件末尾添加

```
[ -f ~/.fzf.zsh ] && source ~/.fzf.zsh
```

或者直接在 `~/.zshrc` 文件中添加配置。

MacOS 上的下载

```
#安装
brew install fzf
#如果要使用内置的快捷键绑定和命令行自动完成功能的话可以按需安装(此处是我安装目录下的
install文件)
/opt/homebrew/Cellar/fzf/0.32.0/install
```

原生使用

单纯的 fzf 命令，可以展示当前目录下的所有文件列表。但除了展示文件列表，fzf 不会做任何事情。

```
fzf
```

fzf 默认会从 STDIN 读入数据，然后将结果输出到 STDOUT

```
find * -type f | fzf > selected
```

上面命令从 find 的搜索结果中读入，输出到文件 selected 中。 如果没有管道连接，fzf 默认使用find来作为搜索工具（可以通过编辑 `FZF_DEFAULT_COMMAND` 重载默认命令）

布局

默认情况下，fzf 以全屏模式启动，但你可以使用 `--height` 选项使它在光标下方以特定高度启动。

```
vim $(fzf --height 40%)
```

当然你也可以使用 `--reverse` 和 `--layout`选项，选择“自顶向下”或“自下而上”的布局。

```
vim $(fzf --height 40% --reverse)
```

您可以将这些选项添加到 `$FZF_DEFAULT_OPTS` 中，以便在默认情况下使用它们。例如：

```
export FZF_DEFAULT_OPTS='--height 40% --layout=reverse --border'
```

搜索语法

在执行 fzf 后，会显示输出交换窗口，在窗口下方的 > 图标后可以输入多个以空格隔开的搜索关键词以实现搜索。如 ^music .mp3$, sbtrkt !fire.

Token	Match type	Description
sbtrkt	fuzzy-match	匹配sbtrkt
`wild	exact-match (quoted)	精确包含wild
^music	prefix-exact-match	以music开头

Token	Match type	Description
.mp3$	suffix-exact-match	以.mp3结尾
!fire	inverse-exact-match	不包含fire
!^music	inverse-prefix-exact-match	不以music开头
!.mp3$	inverse-suffix-exact-match	不以.mp3结尾

如果不想使用模糊匹配，可以使用 `fzf -e` 或者 `fzf --exact` 启动 fzf 实现精确匹配。 可以使用 `|`实现或运算，以形成更为复杂的逻辑。比如：

```
^core go$ | rb$ | py$
```

表示以 core 开头， 以 go，rb 或 py 结尾的文件。

finder 中的快捷键

- `CTRL-K` / `CTRL-J` (or `CTRL-P` / `CTRL-N` or 上/下方向键) 令光标上下移动。
- `Enter` 键选中条目, `CTRL-C` / `CTRL-G` / `ESC` 键退出。
- 在多选模式下 (-m), `TAB` 和 `Shift-TAB` 用于多选。
- 支持 Emacs 式的按键绑定。
- 鼠标: 上下滚动，选中，双击; `shift-click` 或 `shift-scroll` 用于多选模式。

命令行下的快捷键

仅支持在 bash，zsh 和 fish 下使用。需要执行安装目录下的 install 文件 才能使快捷键生效。命令如下（在 Mac 中）：

```
#如果要使用内置的快捷键绑定和命令行自动完成功能的话可以按需安装(此处是我安装目录下的
install文件)
/opt/homebrew/Cellar/fzf/0.32.0/install
```

- `CTRL-T` - 打开 fzf 的输出交换窗口，将选中的文件和路径粘贴至命令行。
    - 设置 `FZF_CTRL_T_COMMAND` 以覆盖默认命令。
    - 设置 `FZF_CTRL_T_OPTS` 以传递其他选项。
        - 你可以设置 `--preview` 选项预览光标选中的文件。

```
# Using highlight (http://www.andre-
simon.de/doku/highlight/en/highlight.html)
export FZF_CTRL_T_OPTS="--preview '(highlight -O ansi -l {} 2>
/dev/null || cat {} || tree -C {}) 2> /dev/null | head -200'"
```

- 可以设置 `--select-1` 和 `--exit-0`。当仅有一个选项时，`--select-1` 会自动选中条目，而不必使用 `Enter` 键；当列表为空时，`--exit-0` 自动退出。

```
export FZF_CTRL_T_OPTS="--select-1 --exit-0"
```

- `CTRL-R` - 打开 fzf 窗口，显示历史命令，选中条目将被拷贝至命令行。
  - 如果想按时间顺序查看命令，请再次按 `CTRL-R` 键，这将切换出按相关性排序。
  - 设置 `FZF_CTRL_R_OPTS` 以传递其他命令。
    - 排序和精确匹配。默认情况下会按时间顺序并禁用排序。如果希望在默认情况下启动按关联性排序或者精确（非模糊）匹配，使用如下命令。

```
export FZF_CTRL_R_OPTS='--sort --exact'
```

    - 预览完整命令。过长的命令在窗口上不完全可见。我们可以使用 `--preview` 选项在预览窗口中显示完整命令。

```
export FZF_CTRL_R_OPTS="--preview 'echo {}' --preview-window
down:3:hidden:wrap --bind '?:toggle-preview'"
```

- `ALT-C` - 打开 fzf 窗口，进入所选目录。
  - 设置 `FZF_ALT_C_COMMAND` 以覆盖默认命令。
  - 设置 `FZF_ALT_C_OPTS` 以传递其他选项。
    - 使用 tree 命令，可以预览选中路径下的目录。

```
export FZF_ALT_C_OPTS="--preview 'tree -C {} | head -200'"
```

命令行下的模糊补全

默认可以通过 `**` + `<TAB>` 来触发模糊补全。 需要执行安装目录下的 install 文件 才能使模糊补全生效。命令见上文。 模糊补全的样式如下：

- `COMMAND [DIRECTORY/][FUZZY_PATTERN]**<TAB>`

具体使用如下：

- 补全文件/路径

```
# 当前路径下的文件
# - 你可以使用TAB键选择多个项目
vim **<TAB>
# 父目录下的文件
vim ../**<TAB>
# 父目录下与fzf匹配的文件
vim ../fzf**<TAB>
# 主目录下的文件
vim ~/**<TAB>
# 当前目录下的文件（单选）
cd **<TAB>
#  ~/github 目录下与fzf匹配的文件
cd ~/github/fzf**<TAB>
```

- 补全进程ID（Process ID）

```
# 可以使用 <TAB>  <Shift-TAB> 选中多个进程
kill -9 **<TAB>
```

- 补全主机名（Host name） 对于 ssh 和 telnet 命令，提供了主机名的模糊补全。名称从 /etc/hosts 和 ~/.ssh/config 中提取。

```
ssh **<TAB>
telnet **<TAB>
```

- 补全环境变量（environment variable）和别名（aliase）

```
unset **<TAB>
export **<TAB>
unalias **<TAB>
```

设置如下：

```
# 使用~~作为模糊补全的默认触发，而非默认值**
export FZF_COMPLETION_TRIGGER='~~'
# fzf命令的选项
export FZF_COMPLETION_OPTS='--border --info=inline'

# 使用fd (https://github.com/sharkdp/fd) 而非默认的查找find
# 用于列出默认路径选项的命令。
# - 函数的第一个参数是开始遍历的基础路径
# - 详细信息参阅GitHub上的源代码(completion.{bash,zsh})
_fzf_compgen_path() {
  fd --hidden --follow --exclude ".git" . "$1"
}

# 使用fd生成目录完成列表
_fzf_compgen_dir() {
  fd --type d --hidden --follow --exclude ".git" . "$1"
}
```

与其他常用命令的连接使用

Homebrew

```
# Install (one or multiple) selected application(s)
# using "brew search" as source input
bip() {
  local inst=$(brew search "$@" | fzf -m)

  if [[ $inst ]]; then
    for prog in $(echo $inst);
    do; brew install $prog; done;
  fi
}
# Update (one or multiple) selected application(s)
bup() {
  local upd=$(brew leaves | fzf -m)

  if [[ $upd ]]; then
    for prog in $(echo $upd);
    do; brew upgrade $prog; done;
  fi
}
# Delete (one or multiple) selected application(s)
bcp() {
  local uninst=$(brew leaves | fzf -m)

  if [[ $uninst ]]; then
    for prog in $(echo $uninst);
    do; brew uninstall $prog; done;
  fi
}
```

autojump

```
fzfj() {
    if [[ "$#" -ne 0 ]]; then
        cd $(autojump $@)
        return
    fi
    cd "$(autojump -s | sort -k1gr | awk '$1 ~ /[0-9]:/ && $2 ~ /^\// { for
(i=2; i<=NF; i++) { print $(i) } }' |  fzf --height 40% --reverse --inline-
info)"
}
```

fasd

```
# fasd & fzf change directory - jump using `fasd` if given argument, filter
output of `fasd` using `fzf` else
fasdf() {
    [ $# -gt 0 ] && fasd_cd -d "$*" && return
    local dir
    dir="$(fasd -Rdl "$1" | fzf -1 -0 --no-sort +m)" && cd "${dir}" ||
return 1
}
```

Git

```
# fbr - checkout git branch
fbr() {
  local branches branch
  branches=$(git --no-pager branch -vv) &&
  branch=$(echo "$branches" | fzf +m) &&
  git checkout $(echo "$branch" | awk '{print $1}' | sed "s/.* //")
}
# fcoc - checkout git commit
fcoc() {
  local commits commit
  commits=$(git log --pretty=oneline --abbrev-commit --reverse) &&
  commit=$(echo "$commits" | fzf --tac +s +m -e) &&
  git checkout $(echo "$commit" | sed "s/ .*//")
}
# fshow - git commit browser
fshow() {
  git log --graph --color=always \
      --format="%C(auto)%h%d %s %C(black)%C(bold)%cr" "$@" |
  fzf --ansi --no-sort --reverse --tiebreak=index --bind=ctrl-s:toggle-sort
\
      --bind "ctrl-m:execute:
                (grep -o '[a-f0-9]\{7\}' | head -1 |
                xargs -I % sh -c 'git show --color=always % | less -R') <<
```

```
'FZF-EOF'
                {}
FZF-EOF"
}
# fcoc_preview - checkout git commit with previews
fcoc_preview() {
  local commit
  commit=$( glNoGraph |
    fzf --no-sort --reverse --tiebreak=index --no-multi \
        --ansi --preview="$_viewGitLogLine" ) &&
  git checkout $(echo "$commit" | sed "s/ .*//")
}
```

tmux

```
# zsh; needs setopt re_match_pcre. You can, of course, adapt it to your own
shell easily.
tmuxkillf () {
    local sessions
    sessions="$(tmux ls|fzf --exit-0 --multi)"  || return $?
    local i
    for i in "${(f@)sessions}"
    do
        [[ $i =~ '([^:]*):.*' ]] && {
            echo "Killing $match[1]"
            tmux kill-session -t "$match[1]"
        }
    done
}
# tm - create new tmux session, or switch to existing one. Works from within
tmux too. (@bag-man)
# `tm` will allow you to select your tmux session via fzf.
# `tm irc` will attach to the irc session (if it exists), else it will
create it.

tm() {
  [[ -n "$TMUX" ]] && change="switch-client" || change="attach-session"
  if [ $1 ]; then
    tmux $change -t "$1" 2>/dev/null || (tmux new-session -d -s $1 && tmux
$change -t "$1"); return
  fi
  session=$(tmux list-sessions -F "#{session_name}" 2>/dev/null | fzf --
exit-0) &&  tmux $change -t "$session" || echo "No sessions found."
}
# fs [FUZZY PATTERN] - Select selected tmux session
#   - Bypass fuzzy finder if there's only one match (--select-1)
#   - Exit if there's no match (--exit-0)
fs() {
  local session
  session=$(tmux list-sessions -F "#{session_name}" | \
    fzf --query="$1" --select-1 --exit-0) &&
  tmux switch-client -t "$session"
}
# ftpane - switch pane (@george-b)
```

```
ftpane() {
  local panes current_window current_pane target target_window target_pane
  panes=$(tmux list-panes -s -F '#I:#P - #{pane_current_path} #
{pane_current_command}')
  current_pane=$(tmux display-message -p '#I:#P')
  current_window=$(tmux display-message -p '#I')

  target=$(echo "$panes" | grep -v "$current_pane" | fzf +m --reverse) ||
return

  target_window=$(echo $target | awk 'BEGIN{FS=":|-"} {print$1}')
  target_pane=$(echo $target | awk 'BEGIN{FS=":|-"} {print$2}' | cut -c 1)

  if [[ $current_window -eq $target_window ]]; then
    tmux select-pane -t ${target_window}.${target_pane}
  else
    tmux select-pane -t ${target_window}.${target_pane} &&
    tmux select-window -t $target_window
  fi
}

# In tmux.conf
# bind-key 0 run "tmux split-window -l 12 'bash -ci ftpane'"
```

Docker

```
# Select a docker container to start and attach to
function da() {
  local cid
  cid=$(docker ps -a | sed 1d | fzf -1 -q "$1" | awk '{print $1}')

  [ -n "$cid" ] && docker start "$cid" && docker attach "$cid"
}
# Select a running docker container to stop
function ds() {
  local cid
  cid=$(docker ps | sed 1d | fzf -q "$1" | awk '{print $1}')

  [ -n "$cid" ] && docker stop "$cid"
}
# Select a docker container to remove
function drm() {
  local cid
  cid=$(docker ps -a | sed 1d | fzf -q "$1" | awk '{print $1}')

  [ -n "$cid" ] && docker rm "$cid"
}
```

Google Chrome

```
# fzfc - browse chrome history
fzfc() {
  local cols sep google_history open
  cols=$(( COLUMNS / 3 ))
  sep='{::}'

  if [ "$(uname)" = "Darwin" ]; then
    google_history="$HOME/Library/Application
Support/Google/Chrome/Default/History"
    open=open
  else
    google_history="$HOME/.config/google-chrome/Default/History"
    open=xdg-open
  fi
  cp -f "$google_history" /tmp/h
  sqlite3 -separator $sep /tmp/h \
    "select substr(title, 1, $cols), url
     from urls order by last_visit_time desc" |
  awk -F $sep '{printf "%-'$cols's  \x1b[36m%s\x1b[m\n", $1, $2}' |
  fzf --ansi --multi | sed 's#.*\(https*://\)#\1#' | xargs $open > /dev/null
2> /dev/null
}
# fzfb - browse chrome bookmarks
fzfb() {
     bookmarks_path=~/Library/Application\
Support/Google/Chrome/Default/Bookmarks

     jq_script='
        def ancestors: while(. | length >= 2; del(.[-1,-2]));
        . as $in | paths(.url?) as $key | $in | getpath($key) | {name,url,
path: [$key[0:-2] | ancestors as $a | $in | getpath($a) | .name?] | reverse
| join("/") } | .path + "/" + .name + "\t" + .url'

    jq -r "$jq_script" < "$bookmarks_path" \
        | sed -E $'s/(.*)\t(.*)/\\1\t\x1b[36m\\2\x1b[m/g' \
        | fzf --ansi \
        | cut -d$'\t' -f2 \
        | xargs open
}
```

---

glances：资源监控工具

参考 此教程 和 GitHub仓库。 虽然我不知道那一堆东西是什么意思，但是五光十色的好好看耶（雾）

> Glances is a cross-platform monitoring tool which aims to present a large amount of monitoring information through a curses or Web based interface. The information dynamically adapts depending on the size of the user interface.

Installing on Ubuntu

```
sudo apt-get install glances
```

Usage

启动 Glances

```
glances
```

输出如下：

```
shouchehendeMBP.lan (Darwin 12.1 64bit)                  Uptime: 7 days,
8:24:17

                CPU      37.9%   MEM     84.3%   SWAP    67.8%   LOAD
8core
CPU  [ 37.9%]   user     27.9%   total   8.00G   total   4.00G   1 min
4.45
MEM  [ 84.3%]   system    3.3%   used    6.74G   used    2.71G   5 min
4.40
SWAP [ 67.8%]                    free    1.26G   free    1.29G   15 min
4.30

NETWORK       Rx/s   Tx/s   TASKS 356 (1645 thr), 354 run, 0 slp, 2 oth
anpi0           0b     0b
anpi1           0b     0b   CPU%   MEM%    PID USER       THR  NI S
ap1             0b     0b   >44.2  2.9   83880 shouchenc  16    0 R Code
Helper
awdl0           0b     0b    0.0   2.5   83785 shouchenc  25    0 R Code
Helper
bridge0         0b     0b    0.1   1.8   80893 shouchenc  21    0 R Code
Helper
en0             0b     0b    0.4   1.4     665 shouchenc  28    0 R Google
Chrom
en1             0b     0b    0.1   1.3    1852 shouchenc  22    0 R Docker
Deskt
en2             0b     0b    112   1.2   44829 shouchenc  22    0 R Google
Chrom
en3             0b     0b    0.0   1.1     644 shouchenc  29    0 R Electron
en4             0b     0b    0.5   1.1   11579 shouchenc  16    0 R Google
Chrom
llw0            0b     0b    0.0   1.1   11746 shouchenc  16    0 R Google
Chrom
lo0             0b     0b    3.1   1.0    1824 shouchenc  8     0 R qemu-
system-
utun0           0b     0b    9.4   1.0     663 shouchenc  13    0 R Terminal
utun1           0b     0b
utun2           0b     0b   High memory consumption
2022-08-04 18:25:52 CST     2022-08-04 18:25:46 (ongoing) - MEM (84.5)
```

可以设置数据刷新的时间间隔（默认为1秒）：

```
glances -t 5
```

你也可以使用 Glances 监控远程系统。要在远程系统上使用它，使用下面的命令：

```
glances -s
```

你会看到类似下面的输出：

```
Define the password for the Glances server
Password:
Password(confirm):
Glances server is running on 0.0.0.0:61209
```

如你所见，Glances 运行在 61209 端口。 现在，到远程机器上执行下面的命令以连接到指定 IP 地址的 Glances 服务器上。假设 192.168.1.10 是你的 Glances 服务器 IP 地址。

```
glances -c -P 192.168.1.10
```

唤起 Glances 后，可以使用如下快捷键：

- m : 按内存占用排序进程
- p : 按进程名称排序进程
- c : 按 CPU 占用率排序进程
- i : 按 I/O 频率排序进程
- a : 自动排序进程
- d : 显示/隐藏磁盘 I/O 统计信息
- f : 显示/隐藏文件系统统计信息
- s : 显示/隐藏传感器统计信息
- y : 显示/隐藏硬盘温度统计信息
- l : 显示/隐藏日志
- n : 显示/隐藏网络统计信息
- x : 删除警告和严重日志
- h : 显示/隐藏帮助界面
- q : 退出
- w : 删除警告记录

htop：互动式进程查看器

参考 此教程，此教程，官网和 GitHub 仓库。

> htop is an interactive text-mode process viewer for Unix systems. It aims to be a better 'top'.

妈耶！又是炫酷的工具（话说是不是和什么东西重复了？），花花绿绿的用来装逼最棒了！！

htop 是Linux系统中的一个互动的进程查看器，一个文本模式的应用程序(在控制台或者X终端中)，需要ncurses。与Linux传统的top相比，htop更加人性化。它可让用户交互式操作，支持颜色主题，可横向或纵向滚动浏览进程列表，并支持鼠标操作。

htop相比较top的优势：

- 可以横向或纵向滚动浏览进程列表，以便看到所有的进程和完整的命令行。
- 在启动上比top 更快。
- 杀进程时不需要输入进程号。
- htop 支持鼠标选中操作（反应不太快）。
- top 已不再维护。

Installing on Ubuntu

```
sudo apt-get install htop
```

Usage

直接唤出进程信息页面：

```
htop
# 实际页面中，应该是彩色显示，但是复制到代码块里好像没法搞成彩色（笑）

# 数字1，2，3，4分别代表CPU处理器/核，下面是一个四核的处理器
# 1，2，3，4右边长条中显示每一个CPU的总用量情况，注意这条上面会有不同的颜色：
# 蓝色：显示低优先级(low priority)进程使用的CPU百分比。
# 绿色：显示用于普通用户(user)拥有的进程的CPU百分比。
# 红色：显示系统进程(kernel threads)使用的CPU百分比。
# 橙色：显示IRQ时间使用的CPU百分比。
# 洋红色(Magenta)：显示Soft IRQ时间消耗的CPU百分比。
# 灰色：显示IO等待时间消耗的CPU百分比。
```

```
# 青色：显示窃取时间(Steal time)消耗的CPU百分比。

    0[|                                     0.4%]   Tasks: 6, 6 thr; 1 running #
我们在计算机上运行的6个任务(tasks)被分解为6个线程(thread)，其中只有1个进程处于运行
(running)状态。
    1[||                                    2.3%]   Load average: 0.08 0.04 0.00
# 三个值是指系统在最后1分钟，最近5分钟和最后15分钟的平均负载 (0.08 0.04 0.00)

    2[||                                    1.1%]   Uptime: 5 days, 03:17:25 #
表示这个系统一共运行了多长的时间，这里一共运行了5天
    3[|                                     0.4%]
  Mem[||||||||||||||||||||          854M/3.84G]
  Swp[                                0K/1024M]

# Mem 和 Swp 两条提供了内存（Memory）和交换（Swap）使用情况。 类似于CPU中的进度条，内
存监视也包含具有多种颜色的进度条：
# 绿色：显示内存页面占用的RAM百分比
# 蓝色：显示缓冲区页面占用的RAM百分比
# 橙色：显示缓存页面占用的RAM百分比

  PID USER     △PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command

    1 root       20   0  155M 21024  6268 S  0.0  0.5  0:00.69
/usr/bin/qemu-x86_64 /bin/zsh /bi
    8 root       20   0  155M 21024  6268 S  0.0  0.5  0:00.00
/usr/bin/qemu-x86_64 /bin/zsh /bi
   91 root       20   0  157M 15640  4028 S  0.0  0.4  0:00.00
/usr/bin/qemu-x86_64 /usr/sbin/ss
   92 root       20   0  157M 15640  4028 S  0.0  0.4  0:00.00
/usr/bin/qemu-x86_64 /usr/sbin/ss
29447 root       20   0  159M 27208  7708 S  0.0  0.7  0:11.46
/usr/bin/qemu-x86_64 /bin/zsh /bi
29453 root       20   0  159M 27208  7708 S  0.0  0.7  0:00.00
/usr/bin/qemu-x86_64 /bin/zsh /bi
31053 root       20   0  152M  9276  4348 T  0.0  0.2  0:00.13
/usr/bin/qemu-x86_64 /usr/bin/box
31055 root       20   0  152M  9276  4348 T  0.0  0.2  0:00.00
/usr/bin/qemu-x86_64 /usr/bin/box
31132 root       20   0  159M 26028  7204 S  0.0  0.6  0:06.60
/usr/bin/qemu-x86_64 /bin/zsh /bi
31138 root       20   0  159M 26028  7204 S  0.0  0.6  0:00.00
/usr/bin/qemu-x86_64 /bin/zsh /bi
32211 root       20   0  152M 13308  5784 R  1.9  0.3  0:00.08
/usr/bin/qemu-x86_64 /usr/bin/hto
32213 root       20   0  152M 13308  5784 S  0.0  0.3  0:00.00
/usr/bin/qemu-x86_64 /usr/bin/hto

# PID – 描述进程的ID号
# USER – 描述进程的所有者（谁跑的）
# PRI – 描述Linux内核查看的进程优先级
# NI – 描述由用户或root重置的进程优先级
# VIR – 它描述进程正在使用的虚拟内存 （virtual memory）
# RES – 描述进程正在消耗的物理内存（physical memory）
# SHR – 描述进程正在使用的共享内存（shared memory）
# S – 描述流程的当前状态 （state）
# CPU% – 描述每个进程消耗的CPU百分比
# MEM% – 描述每个进程消耗的内存百分比
# TIME+ – 显示自流程开始执行以来的时间
# Command –它与每个进程并行显示完整的命令执行 （比如/usr/lib/R）
```

```
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill
F10Quit
```

F1~F10 的功能和对应的字母快捷键：

功能键	快捷键	功能说明
F1	h 或 ?	获取功能键命令帮助
F2	S	设置区域二展示内容，可以设置颜色方案、列等等
F3	/	搜索，可以搜索command列中的信息
F4	\	过滤，可以过滤command列中匹配的进程
F5	t	按照进程树样式展示
F6	< 或 >	排序，根据选择的列排序
F7	[	设置进程优先级，减小优先级
F8	]	设置进程优先级，增加优先级
F9	k	杀死进程，需要先选中进程，然后选择需要发送的信号量
F10	q	退出

命令行参数：

- -C --no-color 使用一个单色的配色方案
- -d --delay=DELAY 设置延迟更新时间，单位秒
- -h --help 显示htop 命令帮助信息
- -u --user=USERNAME 只显示一个给定的用户的过程
- -p --pid=PID,PID… 只显示给定的PIDs
- -s --sort-key COLUMN 依此列来排序
- -v –version 显示版本信息

交互式命令：

- 上下键或PgUP, PgDn 选定想要的进程，左右键或Home, End 移动字段，当然也可以直接用鼠标选定进程；
- Space 标记/取消标记一个进程。命令可以作用于多个进程，例如 "kill"，将应用于所有已标记的进程
- U 取消标记所有进程
- s 选择某一进程，按s:用strace追踪进程的系统调用
- l 显示进程打开的文件: 如果安装了lsof，按此键可以显示进程所打开的文件

- I 倒转排序顺序，如果排序是正序的，则反转成倒序的，反之亦然
- +, - When in tree view mode, expand or collapse subtree. When a subtree is collapsed a "+" sign shows to the left of the process name.
- a (在有多处理器的机器上) 设置 CPU affinity: 标记一个进程允许使用哪些CPU
- u 显示特定用户进程
- M 按Memory 使用排序
- P 按CPU 使用排序
- T 按time+ 使用排序
- F 跟踪进程: 如果排序顺序引起选定的进程在列表上到处移动，让选定条跟随该进程。这对监视一个进程非常有用：通过这种方式，你可以让一个进程在屏幕上一直可见。使用方向键会停止该功能。
- K 显示/隐藏内核线程
- H 显示/隐藏用户线程
- Ctrl-L 刷新
- Numbers PID 查找: 输入PID，光标将移动到相应的进程上

---

httpie

---

http-prompt

---

joe：编辑器！又见编辑器

睡大觉！自己去 官网 翻技术文档去吧您嘞！

Installing on Ubuntu

```
sudo apt-get install joe
```

Usage

> Please Searching-The-Fucking-Web.

---

locales : Linux 语言环境设置

参考此教程 和维基百科 "维基百科") 。 这个严格上来讲不算是命令行工具，但是在配环境的时候碰到了，顺便记录一下。

> locale 是 Linux 系统中多语言环境的设置接口，在 Linux 中，使用 locale 命令来设置和显示程序运行的语言环境，locale 会根据计算机用户所使用的语言，所在国家或者地区，以及当地的文化传统定义一个软件运行时的语言环境。

命名规则

- `<语言>_<地区>.<字符集编码><@修正值>`
- 譬如 `zh_CN.utf8` :
  - zh : 中文
  - CN : 中国大陆地区
  - utf8 : 字符集
- 又譬如 `de_DE.utf-8@euro` :
  - de : 德语
  - DE : 德国
  - utf-8 : 字符集
  - euro : 按照欧洲习惯加以修正

生成区域设置

想要列出所有启用的区域设置，使用如下命令:

```
locale -a
```

在启用某个区域设置前，需要先生成之。 编辑 `/etc/locale.gen` 文件，取消掉对应的注释（注释掉某行则会移除对应的区域设置）。 然后执行以下命令生成 locale :

```
locale-gen
```

例如对于使用美式英语和中国简体的用户:

```
vim /etc/locale.gen
...
#en_SG ISO-8859-1
```

```
en_US.UTF-8 UTF-8
#en_US ISO-8859-1
...
#zh_CN.GBK GBK
zh_CN.UTF-8 UTF-8
#zh_CN GB2312
...
```

编辑完成以后,通过下面的命令生成 Locale：

```
locale-gen
```

设置当前区域

显示正在使用的 Locale 和相关的环境变量。

```
locale

LANG=en_US.utf8 #默认区域设置
LANGUAGE=en_US.utf8 #后备区域设置
LC_CTYPE="en_US.utf8" #
LC_NUMERIC="en_US.utf8" #
LC_TIME="en_US.utf8" #时间和日期格式
LC_COLLATE="en_US.utf8" #排序格式
LC_MONETARY="en_US.utf8" #货币单位格式
LC_MESSAGES="en_US.utf8" #控制程序输出时所使用的语言，主要是提示信息，错误信息，状态信息，标题，标签，按钮和菜单等
LC_PAPER="en_US.utf8" #默认纸张尺寸大小
LC_NAME="en_US.utf8" #姓名格式
LC_ADDRESS="en_US.utf8" #地址格式
LC_TELEPHONE="en_US.utf8" #电话号码格式
LC_MEASUREMENT="en_US.utf8" #度量衡格式
LC_IDENTIFICATION="en_US.utf8" #locale 对自身包含信息的概述
LC_ALL=en_US.utf8 #测试和排除问题
```

查看已经生成的区域设置：

```
localedef --list-archive
```

或者使用 localectl：

```
localectl list-locales
```

设置当前区域

1. 编辑 `/etc/profile` 文件，在文件末尾添加

```
export LC_ALL=zh_CN.utf8
export LANG=zh_CN.utf8
```

并使之生效

```
source /etc/profile
```

2. 编辑 `/etc/default/locale`文件，在文件末尾添加

```
LANG="en_US.UTF-8"
LANGUAGE="en_US:en"
```

并使之生效

```
source /etc/default/locale
```

3. 创建 `/etc/locale.conf` 文件，添加

```
LANG=en_AU.UTF-8
LC_COLLATE=C
LC_TIME=en_DK.UTF-8
```

并使之生效

```
source /etc/locale.conf
```

4. 编辑 `~/.zshrc` 文件（好像还是这个比较好用） 在文件末尾添加

```
export LC_ALL=en_US.utf8
export LANG=en_US.utf8
export LANGUAGE=en_US.utf8
```

并使之生效

```
source ~/.zshrc
```

---

lolcat：彩虹特效

参考此教程和GitHub仓库。 好耶！可以在命令行里看彩虹字力！可以用彩虹字看代码力！

> Rainbows and unicorns! Lolcat 是一个针对 Linux，BSD 和 OSX 平台的工具，它类似于 cat 命令，并为 cat 的输出添加彩虹般的色彩。 Lolcat 主要用于在 Linux 终端中为文本添加彩虹般的色彩。

Installing on Ubuntu

下载前，请确保系统中安装有最新版本的 ruby。

```
wget https://github.com/busyloop/lolcat/archive/master.zip
unzip master.zip
cd lolcat-master/bin
gem install lolcat
```

下载完成后，使用以下命令来查看 lolcat 的版本号：

```
lolocat --version
```

Usage

lolcat的使用格式如下：

```
lolcat lolcat [OPTION]... [FILE]...
```

OPTIONS:

```
-p, --spread=<f>      色彩宽度 (default: 3.0)
-F, --freq=<f>        渐变的频率 (default: 0.1)
```

```
-S, --seed=<i>          色彩的种子（相当于选择颜色）, 0 = random (default: 0)
-a, --animate           开启动画，使渐变色具有动画效果
-d, --duration=<i>      动画的持续时间（次数）,调整单行内容的动画时间 (default: 12)
-s, --speed=<f>         动画播放的速度 ，调整浏览文件时多行内容的播放速度(default:
20.0)
-i, --invert            切换前后台(bg：将一个在后台暂停的命令，变成继续执行；fg：将后台
中的命令调至前台继续运行)?
-t, --truecolor         24位真彩色
-f, --force             Force color even when stdout is not a tty (tty似乎是终端
的代称)
-v, --version           打印版本信息
-h, --help              显示帮助页面
```

用法的样例如下：

```
# 使用lolcat预览文件（相当于彩色版本的cat命令）
lolcat test.c
# 可以加上-a，开启动画，用-d调整动画的时间，譬如
echo "Hello world!"| lolcat -a -d 40
# 使用-p调整色谱的宽度，用-F调整渐变的频率（譬如使颜色跳变）
lolcat test.c -p 20 -F 100
```

当然，更好玩的是通过管道与其他命令连接：

```
ps | lolcat
date | lolcat
ddate | lolcat # 调制的日历
cal | lolcat
figlet "Hello world!" | lolcat # 输出彩色的艺术字
fortune | lolcat # 彩色的随机名言
fortune | cowsay | lolcat # 让彩色牛牛来说格言
sl | lolcat # 彩色的火车飘过!
```

---

lsb-release：显示发行版本信息

参考此教程。 LSB 是 Linux Standard Base 的缩写， lsb_release 命令用来显示 LSB 和特定版本的相关信息。如果使用该命令时不带参数，则默认加上 -v 参数。

Installing on Ubuntu

似乎不用安装，默认就有。 如果没有的话，apt一下就好。

```
sudo apt-get install lsb-release
```

Usage

```
lsb-release -<options>
```

Options:

- -h, --help 查看帮助信息并退出
- -v, --version show LSB modules this system supports
- -i, --id 查看当前发行版的 ID
- -d, --description 显示当前发行版的描述信息
- -r, --release 查看当前发行版的具体版本号
- -c, --codename 查看当前发行版代号
- -a, --all 显示上面所有信息
- -s, --short 显示简短的描述信息

---

multitail：实时监视文件（tail++）

参考官网和此教程。

> MultiTail是个用来实现同时监控多个文档、类似tail命令的功能的软件。 他和tail的区别就是他会在控制台中打开多个窗口，这样使同时监控多个日志文档成为可能
>
> MultiTail follows files in style, it is tail on steroids.

Installing on Ubuntu

```
sudo apt-get update
sudo apt-get install multitail
```

Usage

- 命令格式大抵如下：

  ```
  multitail [-i] file1 [-i] file2
  ```

- 常见用法：

```
# 单窗口横排显示两个文件
multitail  ex1.log ex2.log
# 单窗口竖排显示两个文件
multitail -s 2 ex1.log ex2.log
# 在多个列当中查看多个文件
multitail -s 3 ex1.log ex2.log ex3.log
# 将多个文件合并至一个窗口
multitail ex1.log -I ex2.log
# 查看文件和执行命令，譬如显示 ps 命令的输出，两秒刷新一次（-l 选项允许命令在另一个窗口中执行）
multitail -R 2 -l "ps"
# 过滤文档
multitail -e "ex" ex1.log # 表示MultiTail仅仅显示ex1.log中包含"ex"字符串的行；
multitail -v -e "ssh"-v -e "gnu-pop3d" -e "localhost" ex.log # 表示仅仅显示不包含“ssh”和“gnu-pop3d”，但是包含“localhost”的行。
# 访问其他主机的日志
multitail -l "ssh -t username@host tail -f file"
```

---

mycli

---

ncdu：磁盘空间管理利器

参考 官网。 超好用的磁盘管理神器（不过是不是感觉这个妹妹我曾见过的？）

> ncdu (NCurses Disk Usage) is a curses-based version of the well-known 'du', and provides a fast way to see what directories are using your disk space.

Installing on Ubuntu

```
sudo apt-get install ncdu
```

Usage

- 大略的写法：

```
ncdu [options] dir
```

如：

```
# 查看路径/var的磁盘使用情况，若不指定，则默认为当前路径
ncdu /var

ncdu 2.1.2 ~ Use the arrow keys to navigate, press ? for help
--- /private/var ------------------------------------------------------
-------------------------
    1.0 GiB [############] /vm
. 803.8 MiB [##########   ] /db
. 728.8 MiB [#########    ] /folders
  90.8 MiB [#            ] /log
.  23.5 MiB [             ] /protected
    3.6 MiB [             ] /logs
  244.0 KiB [             ] /.slm
  236.0 KiB [             ] /MobileSoftwareUpdate
  124.0 KiB [             ] /tmp
.  48.0 KiB [             ] /run
  24.0 KiB [            ] /.slmauth
  12.0 KiB [            ] /.slmbackup
    8.0 KiB [             ] /mobile
    4.0 KiB [             ] /sntpd
    4.0 KiB [             ] /msgs
.   0.0   B [             ] /spool
    0.0   B [             ] /rpc
.   0.0   B [             ] /at
    0.0   B [             ] /containers
    0.0   B [             ] /yp
    0.0   B [             ] /personalized_factory
Total disk usage:   2.6 GiB  Apparent size:   2.6 GiB  Items: 7,596
```

- ***OPTIONS***:

选项	功能
-h	显示帮助信息 |
-v	显示ncdu的版本信息
-o *FILE*	将ncdu的信息导出至文件_FILE_
-f *FILE*	加载之前导出的文件信息
--color *SCHEME*	选择ncdu的样式，ff： 禁用颜色， dark： 显示黑色背景， dark-bg： dark的变种，适用于白色背景的终端(默认值)
--ignore-config	忽略所有配置文件

选项	功能
-e, --extended, --no-extended	启用/禁用扩展信息模式。除了通常的文件信息外，还将读取每个文件的所有权、权限和上次修改时间。这将在导出时产生更大的输出文件。使用文件导出/导入功能时，在导出时（确保信息添加到导出中）和导入时（在内存中读取此额外信息），都需要添加此标志。当导入在没有扩展信息的情况下导出的文件时，此标志无效。
--exclude *PATTERN*	排除匹配的文件
-X *FILE*	排除与文件中的模式匹配的所有文件
--include-caches, --exclude-caches	包括缓存/排除缓存
-L, --follow-symlinks, --no-follow-symlinks	包括/排除符号链接
--include-kernfs, --exclude-kernfs	包括/排除内核
--show-hidden, --hide-hidden	显示或隐藏隐藏文件和路径
--sort COLUMN	选择按哪一列的大小排序，选项有disk-usage (the default), name, apparent-size, itemcount 和 mtime
-x	只计算与被扫描路径位于同一文件系统上的文件和路径
--cross-file-system	与上条相反

选项	功能
-0	扫描目录或导入文件时不要给予任何反馈（即扫描的进度条之类的）
-1	扫描目录或导入文件时仅给予一行反馈，譬如 `/Users/shouchenchen/Qt5....elasticnodes-example.png 428116 files / 56.1 GiB`
-2	在扫描目录或导入文件时提供全屏ncurses界面。这是在扫描时提供任何非致命错误反馈的唯一界面。
-q, --slow-ui-updates, --fast-ui-updates	调整ui的刷新频率，默认情况下每秒更新屏幕10次，使用-q或--slow-ui-update后更改为2次
--enable-shell, --disable-shell	Enable or disable shell spawning from the browser. This feature is enabled by default when scanning a live directory and disabled when importing from file.
--enable-delete, --disable-delete	Enable or disable the built-in file deletion feature. This feature is enabled by default when scanning a live directory and disabled when importing from file. Explicitly disabling the deletion feature can work as a safeguard to prevent accidental data loss.
--enable-refresh, --disable-refresh	Enable or disable directory refreshing from the browser. This feature is enabled by default when scanning a live directory and disabled when importing from file.
-r	只读，当只使用一次时，相当于--disable delete的别名，当给出两次时，它还将添加--disableshell的效果
--show-itemcount, --hide-itemcount	显示或隐藏itemcount列
--show-mtime, --hide-mtime	显示或隐藏上次修改时间的列（需要以-e为前提）

选项	功能
--show-graph, --hide-graph	显示或隐藏显示相对尺寸（即含####）的列
--show-percent, --hide-percent	显示或隐藏百分比列
--graph-style *OPTION*	选择相对尺寸列的图形样式，有 half-block 和 eighth-block
--shared-column *OPTION*	Set to off to disable the shared size column for directories, shared (default) to display shared directory sizes as a separate column or unique to display unique directory sizes as a separate column. These options can also be cycled through in the browser with the 'u' key.
--group-directories-first, --no-group-directories-first	将路径排在文件之前
--confirm-quit, --no-confirm-quit	是否在退出ncdu时进行询问
--confirm-delete, --no-confirm-delete	是否对deldete行为进行询问

- ***KEYS***: 进入ncdu界面后的快捷键。

快捷键	功能

快捷键	功能
?	显示help,keys,about三个界面
up,down,j,k	选择路径移动光标
right,enter,l	打开选中的路径
left,<,h	前往父路径（上一层）
n	按文件名排序（按两次取消）
s	按文件大小排序
C	按项目数排序
a	在显示磁盘使用情况和显示外观大小之间切换。
M	按mtime（修改时间）排序，需要以-e为前提
d	删除选中的路径或文件
t	将路径排在文件前面
g	显示百分比和占比的图形与否
u	显示shared/unique size列与否
c	显示子文件/路径数目与否
m	显示修改时间与否
e	显示隐藏/排除文件与否
i	显示当前选中项目的信息
r	刷新当前目录信息
b	在当前路径中生成shell
q	离开

---

neofetch/screenfetch：看起来很厉害的系统信息查看器

参见此教程和 GitHub 仓库。 因为这两个的功能不能说是雷同只能说是一模一样，所以说就仅以 neofetch 为例（好耶摸鱼）。

> Neofetch displays information about your operating system, software and hardware in an aesthetic and visually pleasing way.

Installing on Ubuntu

```
sudo apt-get install neofetch
```

Usage

- 原生使用，显示 ASCII logo 和系统信息：

```
neofetch
                    'c.          username@usernameMBP.lan
                 ,xNMM.          -------------------------------
               .OMMMMo           OS: macOS 12.1 21C52 arm64
               OMMM0,            Host: MacBookPro17,1
     .;loddo:' loolloddol;.      Kernel: 21.2.0
   cKMMMMMMMMMMNWMMMMMMMMMM0:    Uptime: 10 days, 6 hours, 52 mins
 .KMMMMMMMMMMMMMMMMMMMMMMMWd.    Packages: 236 (brew)
 XMMMMMMMMMMMMMMMMMMMMMMMX.      Shell: zsh 5.8
 ;MMMMMMMMMMMMMMMMMMMMMMMM:       Resolution: 1440x900
 :MMMMMMMMMMMMMMMMMMMMMMMM:       DE: Aqua
 .MMMMMMMMMMMMMMMMMMMMMMMX.      WM: Rectangle
 kMMMMMMMMMMMMMMMMMMMMMMMWd.     Terminal: Apple_Terminal
 .XMMMMMMMMMMMMMMMMMMMMMMMMMk    Terminal Font: SFMono-Regular
   .XMMMMMMMMMMMMMMMMMMMMMMMK.    CPU: Apple M1
     kMMMMMMMMMMMMMMMMMMMMMMd     GPU: Apple M1
     ;KMMMMMMMWXXWMMMMMMMk.      Memory: 1596MiB / 8192MiB
       .cooc,.    .,coo:.
```

- 显示特定发行版的 ASCII logo： 格式如：

```
neofetch --ascii_distro <distroname>
```

实例如：

```
neofetch --ascii_distro windows
        ,.=:!!t3Z3z.,                 username@usernameMBP.lan
      :tt:::tt333EE3                  -------------------------------
      Et:::ztt33EEEL @Ee.,      ..,   OS: macOS 12.1 21C52 arm64
      ;tt:::tt333EE7 ;EEEEEEttttt33#   Host: MacBookPro17,1
    :Et:::zt333EEQ. $EEEEEttttt33QL    Kernel: 21.2.0
    it::::tt333EEF @EEEEEttttt33F      Uptime: 10 days, 6 hours, 54 mins
    ;3=*^```"*4EEV :EEEEEttttt33@.     Packages: 236 (brew)
    ,.=::::!t=., ` @EEEEEtttz33QF      Shell: zsh 5.8
```

```
  ;;;;;;;;;zt33)   "4EEEtttji3P*      Resolution: 1440x900
  :t:::::::::tt33.:Z3z..  `` ,..g.      DE: Aqua
  i:::::::::zt33F AEEEtttt::::ztF       WM: Rectangle
;;;;;;;;;;t33V ;EEEttttt::::t3         Terminal: Apple_Terminal
E:::::::::zt33L @EEEtttt::::z3F        Terminal Font: SFMono-Regular
{3=*^```"*4E3) ;EEEtttt:::::tZ`         CPU: Apple M1
            ` :EEEEtttt::::z7          GPU: Apple M1
                "VEzjt:;;z>*`          Memory: 1594MiB / 8192MiB
```

- 显示较小的 logo： 格式如：

```
neofetch --ascii_distro <distroname>_small
```

实例如：

```
neofetch --ascii_distro ubuntu_small
         _    username@usernameMBP.lan
     ---(_)   --------------------------------
_/  ---  \    OS: macOS 12.1 21C52 arm64
(_) |   |     Host: MacBookPro17,1
  \  --- _/   Kernel: 21.2.0
     ---(_)   Uptime: 10 days, 6 hours, 57 mins
              Packages: 236 (brew)
              Shell: zsh 5.8
              Resolution: 1440x900
              DE: Aqua
              WM: Rectangle
              Terminal: Apple_Terminal
              Terminal Font: SFMono-Regular
              CPU: Apple M1
              GPU: Apple M1
              Memory: 1591MiB / 8192MiB
```

- 可以通过编辑配置文件 `~/.config/neofetch/neofetch.conf` 来选择要隐藏的信息（只要将之注释掉就好）：

- 隐藏 logo：

```
neofetch --off
username@usernameMBP.lan
--------------------------------
OS: macOS 12.1 21C52 arm64
Host: MacBookPro17,1
Kernel: 21.2.0
Uptime: 10 days, 12 hours, 48 mins
Packages: 236 (brew)
Shell: zsh 5.8
```

```
Resolution: 1440x900
DE: Aqua
WM: Rectangle
Terminal: Apple_Terminal
Terminal Font: SFMono-Regular
CPU: Apple M1
GPU: Apple M1
Memory: 1593MiB / 8192MiB
```

- 只显示 logo：

```
neofetch -L
                    'c.
                 ,xNMM.
               .OMMMMo
               OMMM0,
     .;loddo:' loolloddol;.
   cKMMMMMMMMMMNWMMMMMMMMMM0:
 .KMMMMMMMMMMMMMMMMMMMMMMMWd.
 XMMMMMMMMMMMMMMMMMMMMMMMX.
 ;MMMMMMMMMMMMMMMMMMMMMMM:
 :MMMMMMMMMMMMMMMMMMMMMMM:
 .MMMMMMMMMMMMMMMMMMMMMMMX.
 kMMMMMMMMMMMMMMMMMMMMMMMWd.
 .XMMMMMMMMMMMMMMMMMMMMMMMMk
   .XMMMMMMMMMMMMMMMMMMMMMK.
     kMMMMMMMMMMMMMMMMMMMd
     ;KMMMMMMMWXXWMMMMMMMk.
       .cooc,.    .,coo:.
```

- 使用自定义图像作为 logo（通过 jp2a 或 caca）：

```
neofetch --jp2a /path/to/image
# ex:

loo..  ..........,,,,,.........;000   root@93eb1da92021
do:..   ......',;;ccc:::::;'.....o00   -----------------
ol;... .....',;:clooolooooool:,'..,O0   OS: Ubuntu 22.04 LTS x86_64
ol,.........',;:cllooooddooolc:;.'k0   Kernel: 5.10.104-linuxkit
oo:........',;,,,,:cloddxxxxolc:.,O0   Uptime: 6 days, 2 hours, 1 min
ddl.......,;;;;;c:::cloddoc:;;,.l00   Packages: 599 (dpkg)
ooc,;,'...,;;:c:::,ccloddlc::ccc:O00   Shell: zsh 5.8.1
oo:;c;;'.,,;;cllccclllllcl;;c::lO00   CPU: (4)
ool:;c:,',;::cloooooooccccloooolllO00   Memory: 859MiB / 3933MiB
dddc:::,,,;::ccoddddl:ccooloddollx00
oooolc:,',;;::cldddddlc:;::;lddollO00
oooooo:;,,;;;::lodolllccccccooolxOOO
loolool,,',;;::cooc:;;;:c:;;:lllddddd
ddddool:,''',;;:colccccccccclllooolll
oooolol:,,,'.',;;clllllccccccccooolll
ooolllo:,,;;;,''',,:::;;;;;::looollll
```

```
ooooooo:',,;:::;,'...''',,,coooollll
ddoool:'',,;;;;;;;,'...,,;;coollllll


neofetch --caca /path/to/image
# ex:
  ::t%:::::::::;;tSS@8@88@%X8%t;t;;8X;t   root@93eb1da92021
  %:;;;;;;;t@888XX888X88@8888@%t%X@t   -----------------
  @:;;;;;;t%@8@8X8%88888888%8888;SXt   OS: Ubuntu 22.04 LTS x86_64
.;X;;;;ttttX888SX88;88XX888.8888tX8;   Kernel: 5.10.104-linuxkit
.:X:t;;%%X%8888X8X88%t.888@8;8@8%8St   Uptime: 6 days, 2 hours, 4 mins
  .SStStttt88@88888S888888S 8888@X;:S   Packages: 599 (dpkg)
.S@88XX;X888X888@@88X888888@88888 %   Shell: zsh 5.8.1
:;S8@888@888X88888888888.8%88888@XSX   CPU: (4)
.S888888S88888X8888X 8.8t8%88X88@S    Memory: 861MiB / 3933MiB
%.SX8@8@8S8@8@88:888%88;%.X8S.8X8XS
.:;%88S88@@8@88S88X:8S888@8X%8;88@XS
  .;.;8888888@88SX%@ 88@88888S8%X 8
.     ;88@8888888 X88S8X88888%8888;8
  .:;%X88888@88@888888888888Xt%%8X
: ::t%X@8X8X@8S@88S88888@8888X8%8%X%
.:::::: %@8X88%X88@8@@@8@8@88@St%%8XX
t  .;;S88888S8SX@888888SX@8S;t8S%SX
:::tt%%8XX8@888S888%tt@8@S88XSS%@@8X
```

- 自定义信息部分的颜色：

```
# 信息面板的部分按顺序排列：标题、@、下划线、副标题、冒号、信息。
neofetch --colors 3 4 5 6 2 1
```

- 与其他命令通过管道连接使用：

```
# lolcat 添加渐变色：
neofetch | lolcat
# 用 fortune | cowsay 替代 logo：
neofetch --ascii "$(fortune | cowsay -W 30)"
 ____________________________    username@usernameMBP.lan
/ It's all in the mind, ya \     --------------------------------
\ know.                    /     OS: macOS 12.1 21C52 arm64
 ----------------------------    Host: MacBookPro17,1
        \   ^__^                  Kernel: 21.2.0
        \  (oo)\_______          Uptime: 10 days, 12 hours, 50 mins
            (__)\       )\/\     Packages: 236 (brew)
                ||----w |        Shell: zsh 5.8
                ||     ||        Resolution: 1440x900
                                 DE: Aqua
                                 WM: Rectangle
                                 Terminal: Apple_Terminal
                                 Terminal Font: SFMono-Regular
                                 CPU: Apple M1
                                 GPU: Apple M1
                                 Memory: 1591MiB / 8192MiB
```

```
# 用 pv 实现动画显示:
neofetch | pv -qL 100 # -qL后的值为动画速度
```

---

node

---

nvm

---

pstree

---

pv

---

ripgrep

---

shellcheck

---

sl

---

sonarqube-lts

---

starship

---

thefuck

tig

tldr

tmux

tree

wrk

ranger

you-get

zoxide