

Multimedia English

2021年9月16日 16:47

Idiomatic:

我马上回来

Just be two ticks | nearly ready

I'll be right back / two ticks

I'll be back soon/shortly/in a sec/in a

jif/jiffy/moment/minute

Unidiomatic:

I'll be back immediately (Too formal)

I'll come back quickly (stay be there)

你只剩260块了

You are down to 260 (Two Sixty)

Left with 260 only (too wordy)

Have only 260 left

Formulaic Sequences

Come on

Don't worry

How are you tonight

Very well

Might as well

Fill up

Bump into (碰到)

End up doing sth

Make a quick call

How did you know

I wouldn't be surprised if

You should ... if ...

Polyword (聚合词, 固化)

Curry favor (拍马屁)

A black sheep (害群之马)

see eye to eye (看法一致)

Get ducks in a row (井井有

条)

Rain cats and dogs

Carry out

Fed up to the back teeth with

you

Lazy around

Collocation (搭配词)

Chicken curry

a black horse (黑马)

Slight/heavy rain

Absolute disgrace

Of sorts

Institutionalized utterance (惯用话语)

I have no excuse

What I did was wrong

I just want to apologize

I feel deeply appreciated

I deserve everything that's happened to me (我自作自受)

He can't even answer

Don't you dare

You buck your bloody ideas up (不积极就别想了)

Sentence builder (句子构建语块)

My point is that X

I think (that) X
Let me start with X

bottle or draft: 瓶装或桶装生啤

barrel/cask 桶

cocktail or shot 大杯或小杯

neat or ice 常温或加冰

Start a tab or keep it open == 最后结总账 bartender 酒保

A4:I don't think so,
everyone has the right
to .If people get to
marry, the proper age
is after 22. This is
because after

ResearchCenter

2021年9月27日 10:50

做人 做事	1.预研究
做成事	2.Report 文献综述
做大事	3.一篇学术论文

第二步

Unicore || 继续优化unicore32

方向: 安全

众核

十年做一事, 够吃一辈子

只争朝夕, 不服韶华
我做我爱, 不要畏难
盛名之下,

Linux command

2021年10月11日 14:30

tar zxvf [文件名.tar.gz] -c ./ 解压到当前文件夹

.tar.gz 和 .tgz

解压: tar zxvf FileName.tar.gz 压缩: tar zcvf FileName.tar.gz
DirName

.gz

解压1: gunzip FileName.gz 解压2: gzip -d FileName.gz 压缩:
gzip FileName

.tar

解包: tar xvf FileName.tar 打包: tar cvf FileName.tar DirName
.bz

解压1: bzip2 -d FileName.bz 解压2: bunzip2 FileName.bz 压缩:
未知

.tar.bz

解压: tar jxvf FileName.tar.bz

.zip

解压: unzip [选项] 压缩包名

来自 <<http://c.biancheng.net/view/782.html>>

mv xxx.txt <dir> 移动

mv before.txt after.txt 重命名

mv /before /after 文件夹移动 (重命名文件夹)

mv /a /b/c

sudo cp -r 文件名 目标文件夹路径

rm xxx.txt 删除文件

rm -rf <dir> 删除文件夹

[Linux 命令大全 | 菜鸟教程 \(runoob.com\)](#)

Dev\$`pip3 --version` -bash: /usr/local/bin/pip3:

/usr/local/opt/python3/bin/python3.6: bad interpreter: No such file
or directory

已经有大量不同的Python安装,以及至少一个您删除的以前的Python安装.
像这样的情况正是为什么不再推荐直接运行pip或pip3的原因,有利于:

`python3 -m pip install <whatever>`

来自 <<https://www.lzys.cc/p/1168399.html>>

为文件增加可执行权限

`chmod +x build/RISCV_FPR128/gem5.opt`

vim全选, 全部复制, 全部删除

全选 (高亮显示): 按esc后, 然后ggvG或者ggVG

全部复制: 按esc后, 然后ggyG

全部删除: 按esc后, 然后dG

解析:

gg: 是让光标移到首行, 在vim才有效, vi中无效

v: 是进入Visual(可视) 模式

G: 光标移到最后一行

选中内容以后就可以其他的操作了, 比如:

d 删除选中内容

y 复制选中内容到0号寄存器

"+y 复制选中内容到+寄存器, 也就是系统的剪贴板, 供其他程序用

IDEA代码一键对齐 `ctrl + alt + L`

VS code

`Shift + Alt + F`

去掉空白: `Ctrl + Shift + J`

vs

部分对齐 (对齐光标所在行或对齐选中代码行): (按两次组合键)

1. `Ctrl+K`

2. `Ctrl+F`

全部代码对齐: 法一: `Ctrl+K+D` (三个键同时按下)

ssh断开，服务器后台继续运行程序

nohup命令

用途：不挂断地运行命令。

语法：nohup Command [Arg ...] [&]

描述：nohup 命令运行由 Command 参数和任何相关的 Arg 参数指定的命令，忽略所有挂断 (SIGHUP)信号。在注销后使用 nohup 命令运行后台中的程序。要运行后台中的 nohup 命令，添加 & (表示“and”的符号)到命令的尾部。

nohup的简单使用

在执行命令时在命令前面加上nohup，然后回车开始运行。

这时你会发现该有的输出其实并没有输出出来，这个时候不要方，这是因为nohup命令将你的所有输出都输出到了当前文件夹下的nohup.out文件中，自己可以使用vim指令进行一个查看。

nohup命令及其输出文件

nohup命令：如果你正在运行一个进程，而且你觉得在退出帐户时该进程还会不会结束，那么可以使用nohup命令。该命令可以在你退出帐户/关闭终端之后继续运行相应的进程。nohup就是不挂起的意思(no hang up)。

该命令的一般形式为：nohup command &

screen命令

1. 直接在命令行键入screen命令

```
$ screen
```

Screen将创建一个执行shell的全屏窗口。你可以执行任意shell程序，就像在ssh窗口中那样。在该窗口中键入exit退出该窗口，如果这是该screen会话的唯一窗口，该screen会话退出，否则screen自动切换到前一个窗口。

2. Screen命令后跟你要执行的程序。

```
$ screen 【后面跟你执行程序的命令】
```

Screen创建一个执行vi test.c的单窗口会话，退出vi将退出该窗口/会话。

3. 以上两种方式都创建新的screen会话。我们还可以在一个已有screen会话中创建新的窗口。在当前screen窗口中键入C-a c，即Ctrl键+a键，之后再按下c键，screen 在该会话内生成一个新的窗口并切换到该窗口。

screen还有更高级的功能。你可以不中断screen窗口中程序的运行而暂时断开（detach）screen会

话，并在随后时间重新连接（attach）该会话，重新控制各窗口中运行的程序。

screen的简单使用

在所要执行的指令前添加screen.然后程序的运行等一切正常。（nohup的输出是定向到了nohup.out文件中，然而screen指令的输出是直接输出到了屏幕上的）

这个时候如果ssh终端断开了连接。我们只需要再次连接服务器然后输入指令

```
screen -ls
```

然后会得到类似下面的结果：

There is a screen on:

27267.pts-19.TITAN-X (09/08/2017 03:49:10 PM) (Detached)

Socket in /var/run/screen/S-huanghailiang.

这里就会显示ssh断开之前的程序，其实断开后程序依然在后台在运行，只是我们这个时候需要将它放到前台来运行。这个时候我们已经通过screen -ls查询到了线程号是27267了，所以我们只需要执行下面的指令即可恢复到前台了。

```
screen -r 27267
```

```
1
```

如果想杀掉终端可以执行

```
kill 27267
```

byobu命令

来自 <<https://blog.csdn.net/darren817/article/details/79807871>>

gem5 learning

2021年10月12日 13:38

Server link: ssh lirong@162.105.89.151 -p 2238

阿里云: ssh lirong@39.106.89.242

接的时候需要转发两个端口

ssh -L 2333:localhost:2333 -L 2322:localhost:2322 lirong@39.106.89.242

然后在浏览器里访问localhost:2333 就可以进入gitlab并上传代码。chipyard-slim

podman run -it --rm -v /home/lirong/testbc:/home/bench chipyard-slim bash

nohup screen tmux

文件传输指令:

scp -r lirong@39.106.89.242:/home/lirong/riscvTools E:\MPRCLab\gem5_fix

gem5_riscv5 fb89e68f44541d63c0e8aaaf5163670c4affbc2815109186a791cc2f7391dbf3

docker cp []
94ae653f9a0ede91c808824d09ebaf18d934fc99716647be197b772525a15140:/home/gem5/tests/MprcCases

使用xshell来操作服务非常方便, 传文件也比较方便。

就是使用rz, sz

首先, 服务器要安装了rz, sz

yum install lrzsz

当然你的本地windows主机也通过ssh连接了linux服务器

运行rz, 会将windows的文件传到linux服务器

运行sz filename, 会将文件下载到windows本地

Build CMD: scons build/RISCV/gem5.opt -j`nproc`

或者

python3 `which scons` build/RISCV/gem5.opt -j`nproc` [-j\$(nproc)]

Run command line: ./build/RISCV/gem5.opt ./configs/example/se.py -c ./test/ --cpu-type DerivO3CPU --caches --mem-type DDR3_1600_8x8 --mem-size 2GB --output=OUTPUT

-i INPUT, --input=INPUT 程序所需的输入, 用 " " 括起来

--output=OUTPUT Redirect stdout to a file.用 " " 括起来

./build/RISCV_ROB256/gem5.opt --stats-file rob256_qsortl_stats.txt ./configs/example/se.py -c ./tests/qsort_large.riscv --cpu-type DerivO3CPU --caches --output=ROB256_QS_OUTPUT

./build/RISCV_GSHARE_8KB/gem5.opt --stats-file gshare_8kb_whetstone_stats.txt ./configs/example/se.py -c ./tests/whetstone.riscv -o 100000 --cpu-type DerivO3CPU --caches --output=OUTPUT

dokcker imagesID: 6a99d3febe7d

Docker container to work Name: gem5_riscv ID: e9450b6e6417

fb89e68f44541d63c0e8aaaf5163670c4affbc2815109186a791cc2f7391dbf3

创建容器:

docker run -it --rm -v /home/lirong/gem5:/home/gem5 6a99d3febe7d bash

启动容器: docker start gem5_lirong

运行: docker attach gem5_lirong 或 docker exec -it [containerName] /bin/bash

Gcc riscv 工具链版本, Python配置, 用基准测试程序得到数据报告(详细), 分析一下, 下周

传输文件:

得到

得到容器全称: docker inspect -f '{{.Id}}' [Container]
docker cp 本地文件路径 ID全称:容器路径

docker cp coremark-1.01.tar.gz [ID全称]:/home/gem5/tests/MprcCases

riscv编译命令

riscv64-unknown-elf-gcc -static -O2 -o tests/rongl.riscv tests/ronglonely.c -lm

riscv64-unknown-elf-gcc -static whetstone.c -O0 -o whetstone.riscv -lm (不优化版本)

tmux:

Gem5 se options

Options:

-h, --help show this help message and exit
-c CMD, --cmd=CMD The binary to run in syscall emulation mode.
-o OPTIONS, --options=OPTIONS The options to pass to the binary, use " " around the entire string
-i INPUT, --input=INPUT Read stdin from a file.
--output=OUTPUT Redirect stdout to a file.
--errout=ERROUT Redirect stderr to a file.
--ruby
-d, --detailed
-t, --timing
--inorder
-n NUM_CPUS, --num-cpus=NUM_CPUS
--caches
--l2cache
--fastmem
--clock=CLOCK
--num-dirs=NUM_DIRS
--num-l2caches=NUM_L2CACHES
--l1d_size=L1D_SIZE
--l1i_size=L1I_SIZE
--l2_size=L2_SIZE
--l1d_assoc=L1D_ASSOC
--l1i_assoc=L1I_ASSOC
--l2_assoc=L2_ASSOC

```
docker cp
fb89e68f44541d63c0e8aaaf5163670c4affbc2815109186a791cc2f7391dbf3:/home/gem
5/test/ /home/lirong/gem5/testShr
```

docker cp ID全称:容器文件路径 本地路径

查看riscv gnu版本:

```
riscv64-unknown-elf-gcc --version
:: riscv64-unknown-elf-gcc (GCC) 9.2.0
riscv64-unknown-linux-gnu-gcc --version
:: riscv64-unknown-linux-gnu-gcc (GCC) 9.2.0
编译使用: riscv64-unknown-elf-gcc -o [可执行
文件名] [xxx.x源程序名]
```

spike pk [-m128] [可执行文件名] ;spike模拟器执行文件

```
qemu-system-riscv64 --version
:: QEMU emulator version 5.0.0 (v5.0.0-dirty)
```

```
spike --help
Spike RISC-V ISA Simulator 1.0.1-dev
```

```
podman run -it --rm -v BOOM目
录:/root/chipyard/generators/boom/src/main/scala
-v 测试程序目录:/root/tests chipyard-slim bash
```

当前用户有效:

```
export PATH=$PATH:/root/riscv/riscv-tools-
install/bin配置环境变量 ~/.bashrc (等号两边没空
格) 生效: source ~/.bashrc
```

通过修改profile文件: (profile文件在/etc目录下)

```
vi /etc/profile //编辑profile文件
//在最后一行添上: (或在PATH=/.....中加入
"./xxx/xxx")
```

```
export PATH=$PATH:/xxx/xxx
```

生效方法: 系统重启

有效期限: 永久有效

用户局限: 对所有用户

echo \$PATH //查看当前PATH的配置路径

```
echo $LD_LIBRARY_PATH
```

```
export LD_LIBRARY_PATH=
$LD_LIBRARY_PATH:/root/riscv/riscv-tools-
install/lib:/root/riscv/riscv-tools-install/libexec
对于由普通用户自己编译生成的.so库文件, 比较
好的做法是把这些.so库文件的lib路径用export指
令加入到~/.bash_profile中的LD_LIBRARY_PATH变
量中,LD_LIBRARY_PATH是程序运行需要链接.so库
时会去查找的一个目录, ~/.bash_profile是登陆或
打开shell时会读取的文件, 这样, 每次用户登录
时, 都会把这些.so库文件的路径写入
LD_LIBRARY_PATH, 这样就可以正常地使用这
些.so库文件了
```

修改O3CPU的参数, 这里不确定的配置项是
numFetchBufferEntries 16 是否和
fetchQueueSize对应
还有完全不能确定


```

new Config(site, here, up) => {
  case TileAllocated(inBusSystem) => {
    val prev = up(TileAllocated(inBusSystem), site)
    val ldoOffset = overrideIdOffset.getOrElse(prev.size)
    (0 until n).map { i =>
      RocketTileParams {
        tileParams = RocketTileParams {
          core = RocketCoreParams {
            fetchWidth = 4,
            decodeWidth = 2,
            numLdEntries = 64,
            issueParams = Seq(
              IssueParams(issueWidth=1, numEntries=12, iqType=IQ_PPW_11Value, dispatchWidth=2),
              IssueParams(issueWidth=2, numEntries=20, iqType=IQ_ZNT_11Value, dispatchWidth=2),
              IssueParams(issueWidth=1, numEntries=16, iqType=IQ_PP_11Value, dispatchWidth=2)),
            numIntPhysRegisters = 80,
            numGpPhysRegisters = 64,
            numLdEntries = 10,
            numStgEntries = 10,
            numBrCount = 12,
            numStgBufferEntries = 30,
            Ftu = FtuParameters(nEntries=32),
            nPerfCounters = 6,
            fpu = Some(freechips.rocketchip.tile.FPUParams(sfnLatency=4, dfmtLatency=4, divSqrt=true))
          },
          dCache = Some(
            DCacheParams(rowBits = site(SystemBusKey).beatBits, nSets=64, mways=4, nSMBs=2, nL1Mays=8)
          ),
          iCache = Some(
            ICacheParams(rowBits = site(SystemBusKey).beatBits, nSets=64, mways=4, fetchBytes=2*4)
          ),
          hartId = i + ldoOffset
        },
        crossingParams = RocketCrossingParams()
      }
    } ++ prev
  }
  case SystemBusKey => up(SystemBusKey, site).copy(beatBytes = 8)
  case Xlen => 64
}

```

Docker用法

2021年10月18日 17:58

【拉取提交推送镜像】

登录阿里云Docker Registry:

`docker login --username=lirongqbjs registry.cn-hangzhou.aliyuncs.com`

Password: lr991004

从阿里云Registry拉取镜像: `docker pull registry.cn-hangzhou.aliyuncs.com/ronglonely/gem5:latest`

`docker commit -m "<提交时的说明文字>" -a "ronglonely <lonelystrange@foxmail.com>" [containID] registry.cn-hangzhou.aliyuncs.com/ronglonely/gem5:latest`

重命名: `docker tag [ImageID] registry.cn-hangzhou.aliyuncs.com/ronglonely/gem5:latest` ([REPOSITORY:[TAG]] 镜像源名+新的版本)

将镜像推送到Registry: `docker push registry.cn-hangzhou.aliyuncs.com/ronglonely/gem5:latest`

【镜像操作】

查看所有镜像: `docker images`

为none tag的镜像重命名 tag标识: `docker tag [IMAGE ID] [REPOSITORY名]:[TAG标识]`

`docker`从镜像创建容器: `docker run -it [Image] [COMMAND]` (eg:/bin/bash, shell的程序路径, bash是一种shell, 如sh)

`--name <nginx-lb>` (为容器指定一个别名)

`--rm`: 容器退出时自动清理容器内部的文件系统。

`-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

eg:-p 127.0.0.1:80:8080/tcp (绑定容器的 8080 端口, 并将其映射到本地宿主机 127.0.0.1 的 80 端口上)

在大部分的场景下, 我们希望 docker 的服务是在后台运行的, 我们可以过 `-d` 指定容器的运行模式。

`docker run -itd --name [重命名] [Image] /bin/bash` (加了 `-d` 参数默认不会进入容器, 想要进入容器需要使用指令 `docker exec/attach [containerID/NAME] -v /src:/dst` 绑定卷)

【文件共享】

`docker run -it --name gem5test -v /home/gem5test:/gem 6a99d3febe7d bash`

挂载本地目录 (绑定卷) : `docker run --name [rename] -it -v /opt:/soft [Image] /bin/bash` (这样启动后容器会自动在根目录下创建 soft(docker的)文件夹, 同时也就要求了**/soft必须写绝对路径**, :ro表示容器内只读)

`docker run -it --privileged=true -v /test:/soft [Image] /bin/bash` (以特权方式启动容器, 在容器内操作宿主机已有目录, 不再报 Permission denied) 或者在宿主机运行setenforce 0 关闭selinux

用于容器与主机之间的数据拷贝:

`docker cp [OPTIONS] CONTAINER:SRC_PATH DEST_PATH|-`

`docker cp [OPTIONS] SRC_PATH|- CONTAINER:DEST_PATH`

OPTIONS说明:

`-L`:保持源目标中的链接

实例

将主机/www/runoob目录拷贝到容器96f7f14e99ab的/www目录下

`docker cp /www/runoob 96f7f14e99ab:/www/`

将容器96f7f14e99ab的/www目录拷贝到主机的/tmp目录中

`docker cp 96f7f14e99ab:/www /tmp/`

检查容器里文件结构的更改:

`docker diff [OPTIONS] CONTAINER`

实例

查看容器mymysql的文件结构更改。

`docker diff mymysql`

【镜像之上的容器操作】

查看所有容器：`docker ps -a`

启动容器：`docker start [containerID/NAME]`

停止容器：`docker stop [containerID/NAME]`

重启容器：`docker restart [containerID/NAME]`

进入容器：`docker attach/exec [containerID/NAME]` (其中exec最后面需要加COMMAND (一般是`docker exec -it [containerID/NAME] /bin/bash`) , 如果从这个容器退出, 容器不会停止)

导出容器：`docker export [containerID/NAME] > [filepath/name]`

导入容器：`docker import file|URL - [REPOSITORY[:TAG]]`

删除容器：`docker rm -f [containerID/NAME]`

删除镜像：`docker rmi [Image]` (先删除容器, 再删除镜像)

查看容器DSN信息：`cat etc/resolv.conf`

重启docker：`/etc/init.d/docker restart`

【更改镜像】

当我们从 docker 镜像仓库中下载的镜像不能满足我们的需求时, 我们可以通过以下两种方式对镜像进行更改。

1、从已经创建的容器中更新镜像, 并且提交这个镜像

2、使用 Dockerfile 指令来创建一个新的镜像

更新镜像

更新镜像之前, 我们需要使用镜像来创建一个容器。

在运行的容器内, 安装编译工具链或使用 `apt-get update` 命令进行更新。

在完成操作之后, 输入 `exit` 命令来退出这个容器。

此时 ID 为 [containID(见命令行批头)] 的容器, 是按我们的需求更改的容器。我们可以通过命令 `docker commit` 来提交容器副本。

提交容器副本, 打包到镜像:

`docker commit -m="<提交时的说明文字>" -a="ronglonely <lonelystrange@foxmail.com>" [containID] [REPOSITORY[:TAG]]`

各个参数说明:

-m: 提交的描述信息

-a: 指定镜像作者

-c :使用Dockerfile指令来创建镜像;

[containID]: 容器 ID

[REPOSITORY[:TAG]]: 指定要创建的目标镜像名

我们可以使用 `docker images` 命令来查看我们的新镜像:

再使用如下cmd将镜像推送到阿里云Registry共享

`docker tag [ImageId] registry.cn-hangzhou.aliyuncs.com/ronglonely/gem5:latest`

`docker push registry.cn-hangzhou.aliyuncs.com/ronglonely/gem5:latest`

构建镜像

我们使用命令 `docker build` , 从零开始来创建一个新的镜像。为此, 我们需要创建一个 Dockerfile 文件, 其中包含一组指令来告诉 Docker 如何构建我们的镜像。

```
runoob@runoob:~$ cat Dockerfile
```

```
FROM centos:6.7
```

```
MAINTAINER Fisher "fisher@sudops.com"
```

```
RUN /bin/echo 'root:123456' |chpasswd
```

```
RUN useradd runoob
```

```

RUN /bin/echo 'runoob:123456' |chpasswd
RUN /bin/echo -e "LANG=\"en_US.UTF-8\"" >/etc/default/locale
EXPOSE 22
EXPOSE 80
CMD /usr/sbin/sshd -D

```

每一个指令都会在镜像上创建一个新的层，每一个指令的前缀都必须是大写的。

第一条FROM，指定使用哪个镜像源

RUN 指令告诉docker 在镜像内执行命令，安装了什么。。。

然后，我们使用 Dockerfile 文件，通过 docker build 命令来构建一个镜像。

```

runoob@runoob:~$ docker build -t runoob/centos:6.7 .
Sending build context to Docker daemon 17.92 kB
Step 1 : FROM centos:6.7
---> d95b5ca17cc3
Step 2 : MAINTAINER Fisher "fisher@sudops.com"
---> Using cache
---> 0c92299c6f03
Step 3 : RUN /bin/echo 'root:123456' |chpasswd
---> Using cache
---> 0397ce2fbd0a
Step 4 : RUN useradd runoob

```

.....
参数说明：

-t：指定要创建的目标镜像名

.：Dockerfile 文件所在目录，可以指定Dockerfile 的绝对路径

使用docker images 查看创建的镜像已经在列表中存在,镜像ID为860c279d2fec

```

runoob@runoob:~$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
runoob/centos        6.7                860c279d2fec       About a minute ago 190.6 MB
runoob/ubuntu        v2                 70bf1840fd7c       17 hours ago       158.5 MB
ubuntu              14.04             90d5884b1ee0       6 days ago         188 MB
php                  5.6               f40e9e0f10c8       10 days ago        444.8 MB
nginx                latest            6f8d099c3adc       12 days ago        182.7 MB
mysql                5.6               f2e8d6c772c0       3 weeks ago        324.6 MB
httpd                latest            02ef73cf1bc0       3 weeks ago        194.4 MB
ubuntu               15.10             4e3b13c8a266       5 weeks ago        136.3 MB
hello-world          latest            690ed74de00f       6 months ago        960 B
centos                6.7               d95b5ca17cc3       6 months ago       190.6 MB
training/webapp      latest            6fae60ef3446       12 months ago      348.8 MB

```

我们可以使用新的镜像来创建容器

```

runoob@runoob:~$ docker run -t -i runoob/centos:6.7 /bin/bash
[root@41c28d18b5fb /]# id runoob
uid=500(runoob) gid=500(runoob) groups=500(runoob)

```

从上面看到新镜像已经包含我们创建的用户 runoob。

gem5_riscv tests

2021年10月22日 13:05

基准程序多参数情况:

```
./build/RISCV/gem5.opt --stats-file mean_susan_stats.txt ./configs/example/se.py -c ./exam/susan -o
"/exam/mibench-riscv/automotive/susan/input_large.pgm" ./exam/output_small.smoothing.pgm -s" --
cpu-type DerivO3CPU --caches --output=MEAN_SUSAN_OUTPUT
```

```
Whetstone:
root@680e9f95c5d:/home/whetstone# spike pk whetstone_riscv
bbl loader

Insufficient duration- Increase the LOOP count
root@680e9f95c5d:/home/whetstone# spike pk whetstone_riscv 100000
bbl loader

Loops: 100000, Iterations: 1, Duration: 21 sec
C Converted Double Precision Whetstones: 476.2 MIPS
root@680e9f95c5d:/home/whetstone#
```

```
Dhrystone
IPR 40:
Dhrystone(1.1) time for 50000 passes = 172
This machine benchmarks at 289 dhrystones/second
root@14967a8c04bd:/home/gem5# cat nSout/IPR160_DHRY_OUTPUT
Dhrystone(1.1) time for 50000 passes = 172
This machine benchmarks at 289 dhrystones/second
```

```
----- Begin Simulation Statistics -----
simSeconds          0.017419          # Number of seconds
simulated (Second)
simTicks            17419395000        # Number of ticks
simulated (Tick)
finalTick           17419395000        # Number of ticks from
beginning of simulation (restored from checkpoints and never reset) (Tick)
simFreq            1000000000000       # The number of ticks
per simulated second ((Tick/Second))
hostSeconds         257.47             # Real time elapsed on
the host (Second)
hostTickRate        67655922           # The number of ticks
simulated per host second (ticks/s) ((Tick/Second))
hostMemory          687848             # Number of bytes of
host memory used (Byte)simlnsts
Number of instructions simulated (Count)
simOps              36858006           # Number of ops
(including micro ops) simulated (Count)
hostInstRate        143154             # Simulator instruction
rate (inst/s) ((Count/Second))
hostOpRate          143154             # Simulator op (including
micro ops) rate (op/s) ((Count/Second))
system_clk_domain.clock 1000          # Clock period in ticks
(Tick)
system.cpu.numCycles 34838791          # Number of cpu
cycles simulated (Cycle)
system.cpu.numWorkItemsStarted 0        # Number of work
items this cpu started (Count)
system.cpu.numWorkItemsCompleted 0      # Number of
work items this cpu completed (Count)
system.cpu.instsAdded 43314865         # Number of
instructions added to the IQ (excludes non-spec) (Count)
```

```
IPR 160:
root@14967a8c04bd:/home/gem5# cat nSout/IPR160_DHRY_OUTPUT
Dhrystone(1.1) time for 50000 passes = 172
This machine benchmarks at 289 dhrystones/second
stats:
----- Begin Simulation Statistics -----
simSeconds          0.000002          # Number of seconds simulated (Second)
simTicks            2299000           # Number of ticks simulated (Tick)
finalTick           2299000           # Number of ticks from beginning of
simulation (restored from checkpoints and never reset) (Tick)
simFreq            1000000000000       # The number of ticks per simulated
second ((Tick/Second))
hostSeconds         0.01              # Real time elapsed on the host (Second)
hostTickRate        400342843          # The number of ticks simulated per host
second (ticks/s) ((Tick/Second))
hostMemory          680944             # Number of bytes of host memory used
(Byte)simlnsts
Number of instructions simulated
simOps              3603              # Number of ops (including micro ops)
simulated (Count)
hostInstRate        612659            # Simulator instruction rate (inst/s)
((Count/Second))
hostOpRate          611219            # Simulator op (including micro ops) rate
(op/s) ((Count/Second))
system_clk_domain.clock 1000          # Clock period in ticks (Tick)
system.cpu.numCycles 4599             # Number of cpu cycles simulated
(Cycle)
system.cpu.numWorkItemsStarted 0        # Number of work items this cpu
started (Count)
system.cpu.numWorkItemsCompleted 0      # Number of work items this cpu
completed (Count)
system.cpu.exec_context.thread_0.numlnsts 3603 # Number of instructions
committed (Count)
system.cpu.exec_context.thread_0.numOps 3603 # Number of ops (including
micro ops) committed (Count)
```

```
IPR 240:
root@680e9f95c5d:/home/whetstone# spike pk whetstone_riscv
bbl loader

Insufficient duration- Increase the LOOP count
root@680e9f95c5d:/home/whetstone# spike pk whetstone_riscv 100000
bbl loader

Loops: 100000, Iterations: 1, Duration: 21 sec
C Converted Double Precision Whetstones: 476.2 MIPS
root@680e9f95c5d:/home/whetstone#
```

```
----- Begin Simulation Statistics -----
simSeconds          0.000002          # Number of seconds simulated (Second)
simTicks            2299000           # Number of ticks simulated (Tick)
finalTick           2299000           # Number of ticks from beginning of
simulation (restored from checkpoints and never reset) (Tick)
simFreq            1000000000000       # The number of ticks per simulated
second ((Tick/Second))
hostSeconds         0.01              # Real time elapsed on the host (Second)
hostTickRate        400342843          # The number of ticks simulated per host
second (ticks/s) ((Tick/Second))
hostMemory          680944             # Number of bytes of host memory used
(Byte)simlnsts
Number of instructions simulated
simOps              3603              # Number of ops (including micro ops)
simulated (Count)
hostInstRate        612659            # Simulator instruction rate (inst/s)
((Count/Second))
hostOpRate          611219            # Simulator op (including micro ops) rate
(op/s) ((Count/Second))
system_clk_domain.clock 1000          # Clock period in ticks (Tick)
system.cpu.numCycles 4599             # Number of cpu cycles simulated
(Cycle)
system.cpu.numWorkItemsStarted 0        # Number of work items this cpu
started (Count)
system.cpu.numWorkItemsCompleted 0      # Number of work items this cpu
completed (Count)
system.cpu.exec_context.thread_0.numlnsts 3603 # Number of instructions
committed (Count)
```

```
Dhrystone 500000:
IPR40:
----- Begin Simulation Statistics -----
simSeconds          0.164800          # Number of seconds simulated (Second)
simTicks            164800193500       # Number of ticks simulated (Tick)
finalTick           164800193500       # Number of ticks from beginning of simulation
(restored from checkpoints and never reset) (Tick)
simFreq            1000000000000       # The number of ticks per simulated second
((Tick/Second))hostSeconds 2379.94    # Real time elapsed on the host
(Second)
hostTickRate        69245548          # The number of ticks simulated per host second
(ticks/s) ((Tick/Second))
```

```
riscv64-unknown-elf-objdump -d hello.o/hello_riscv -M no-
aliases,numeric > hello.s
```

```
Mibench:
qsort_large :
ROB256
system.cpu.ipc 0.138665
```

```
basicmath_large :
ROB256:
```

```
sha_large:
```

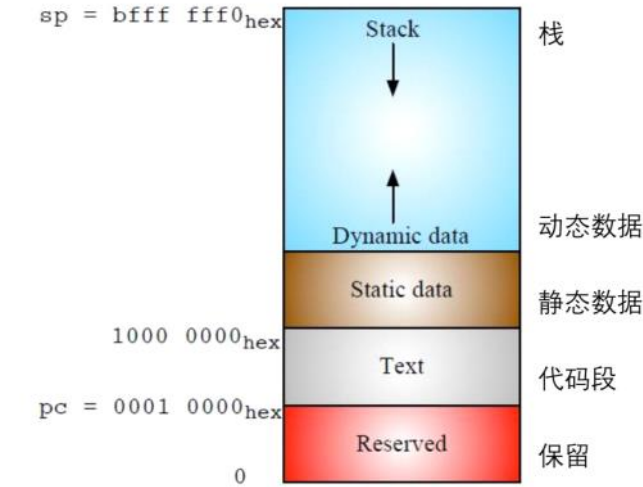
```
dijkstra_large:
```

```
bitcnts:
```

```
reorder buffer entries
```

```
docker run -it --rm -v /home/lirong/gem5:/home/gem5 6a99d3febe7d bash
podman run -it --rm -v /home/lirong/testbc:/home/gem5 6a99d3febe7d bash
```

RV32I 为程序和数据分配内存。图中的顶部是高地址，底部是低地址。在 RISC-V 软件规范中，栈指针 (sp) 从 0xbffffff0 开始向下增长；程序代码段从 0x00010000 开始，包括静态链接库；程序代码段结束后是静态数据区，在这个例子中假设从 0x10000000 开始；然后是动态数据区，由 C 语言中的 malloc() 函数分配，向上增长，其中包含动态链接库。



来自 <<https://www.cnblogs.com/mikewolf2002/p/11323406.html>>

```
./build/RISCV_IWB/gem5.opt --stats-file iw8_susan_stats.txt ./configs/example/se.py -c ./exam/susan -o "/exam/mibench-riscv/automotive/susan/input_large.pgm" ./exam/out_large.corners.pgm -c" --cpu-type O3CPU --caches
```


iterated over during squash; mainly for profiling (Count)

40:

载人航天专题讲座

2021年10月22日 18:43 陈善广老师

嫦娥奔月

万户飞天

敦煌飞天

飞天是人类自古以来不断追求的梦想，更是中华民族的千年梦想。

1965年，毛泽东主席登上井冈山，写诗说道，可上天九天揽月，可下五洋捉鳖，坐地日行八万里，巡天遥看一千河

2003，毛主席的预言成真了。

太空被称为陆海空之外的“第四疆域”，刷新着对自然的认识，推动科技进步与经济发展阿波罗计划有着对美国的科技进步%50以上的作用，无预算计划，经济代价太大，科学价值在哪。

美苏争霸，争第一，意识形态问题。

维护国家安全，促进综合国力 25公里以外不属于任何国家，没有领天
提高国际地位，增强民族凝聚力。华人奶奶对中国飞天第一人杨利伟说道，
你们飞的有多高，我们海外华人的头就能昂多高

正是这种

世界载人航天，从1961年前苏联加加林乘坐东方一号，揭开序幕。1969年，载人登月。第一个空间站前苏联礼炮号航天站，俄罗斯第三代空间站-和平号空间站 1998年开始国际空间站

中国载人航天工程网

美国平民载人航天，最年长的航天员90多岁，已经上天了。地月空间站，阿特米斯计划，第一位女航天员送上月球，以突破技术为目的，值得学习

总设计师期望10年内会登上月球。神舟十三号是核心技术验证的收官之战。
神舟十四号 开始空间站的构建，构成丁字型，十五号，实现在轨交接班技术。

无人探火星 有一半都夭折在路上，250多天。

载人航天基本技术，主要目的是为了发展空间科学与应用，造福人类。

载人航天飞机一次性可运载7个人，类似空天飞机，可重复使用未达到目标，维修费用和重建费用接近，系统过于复杂，安全性低，控制论难以解决问题。技术积累值得

谁救了NASA，埃隆马斯克，短时间实现龙飞船，真实版钢铁侠，全能型人才，世界首富。

45天以后身体机能达到稳态。

2个月以上叫长期在轨生存。

聂海胜的使命感精神值得学习。

载人登火是未来三十年的远景目标。

国家自然科学基金委交叉学部的成立，科技与人文的融合

Tmux快捷操作

2021年10月25日 18:58

tmux/screen rz/sz 文件卡死快速退出方法(tmux 暂不能使用rz/sz)
按住Ctrl键, 再按五次x键 (强行终断传输)

查看所有tmux会话

指 令: tmux ls

快捷键: Ctrl+b s

新建tmux窗口

指 令: tmux new -s <session-name>

重命名会话

指 令: tmux rename-session -t <old-name> <new-name>

快捷键: Ctrl+b \$

分离会话

指 令: tmux detach 或者使用 exit(关闭窗口)

快捷键: Ctrl+b d

重新连接会话

指 令: tmux attach -t <session-name> 或者使用 tmux at -t <session-name>

#平铺当前窗格 (个人很喜欢的快捷键, 注意: 平铺的是当前选中的窗格)

快捷键: Ctrl+b z (再次 Ctrl+b z 则恢复)

杀死会话

指 令: tmux kill-session -t <session-name>

切换会话

指 令: tmux switch -t <session-name>

划分上下两个窗格

指 令: tmux split

快捷键: Ctrl+b “

划分左右两个窗格

指 令: tmux split -h

快捷键: Ctrl+b %

光标切换到上方窗格

指 令: tmux select-pane -U

快捷键: Ctrl+b 方向键上

光标切换到下方窗格

指 令: tmux select-pane -D

快捷键: Ctrl+b 方向键下

光标切换到左边窗格

指 令: tmux select-pane -L

快捷键: Ctrl+b 方向键左

光标切换到右边窗格

指 令: tmux select-pane -R

快捷键: Ctrl+b 方向键右

面板操作 (ctrl + b后生效)

” 将当前面板平分为上下两块

% 将当前面板平分为左右两块

x 关闭当前面板

! 将当前面板置于新窗口; 即新建一个窗口, 其中仅包含当前面板

Ctrl+方向键 以1个单元格为单位移动边缘以调整当前面板大小

Alt+方向键 以5个单元格为单位移动边缘以调整当前面板大小

Space 在预置的面板布局中循环切换; 依次包括even-horizontal、even-vertical、main-horizontal、main-vertical、tiled

q 显示面板编号

o 在当前窗口中选择下一面板

方向键 移动光标以选择面板

{ 向前置换当前面板

} 向后置换当前面板

Alt+o 逆时针旋转当前窗口的面板

Ctrl+o 顺时针旋转当前窗口的面板

窗口操作

c 创建新窗口

& 关闭当前窗口

数字键 切换至指定窗口

p 切换至上一窗口

n 切换至下一窗口

l 在前后两个窗口间互相切换

w 通过窗口列表切换窗口

, 重命名当前窗口; 这样便于识别

. 修改当前窗口编号; 相当于窗口重新排序

f 在所有窗口中查找指定文本

系统操作

? 先按下Ctrl+b, 再按下?, 显示帮助信息,列出所有快捷键; 按q返回

d 脱离当前会话; 这样可以暂时返回Shell界面, 输入tmux attach能够重新进入之前的会话

D 选择要脱离的会话; 在同时开启了多个会话时使用

Ctrl+z 挂起当前会话

r 强制重绘未脱离的会话

s 选择并切换会话; 在同时开启了多个会话时使用

: 进入命令行模式; 此时可以输入支持的命令, 例如kill-server可以关闭服务器

[进入复制模式; 此时的操作与vi/emacs相同, 按q/Esc退出

~ 列出提示信息缓存; 其中包含了之前tmux返回的各种提示信息

Shell操作

2021年10月25日 22:44

查找文件中包含关键字的行

grep --color=auto [option] '[key]' [file]

option:

-n 带行号

-l 忽略大小写

-w 按单词搜索

key=^str 以str开头的行, str\$ 以str结尾 '^\$' 空行

-v 取反

-B n 前n行 -A n 再包括后n行 -C n 前后n行 'error'

cat -n [file] 带行号查看文件内容

cut -d: -f1,7 [file] 截取文件中以冒号分割的第1和7列

-c1-5 截取每列第1到5个字符 -c10- 从10到后全部截取

| 管道符号, 前一个结果用到后面

tail -1 [file] 截取文件最后一行

sort -nr t: -k3 [file] 以:分割, 第3列的行, 按数字排序, r 表示降序排列 -u 去除重复行 (包括不连续的行)

uniq -c [file] 去除连续重复行, -d

echo hello world | tee -a [file] 从标准输入读取并以附加方式写入到标准输入和文件

diff -c [file1] [file2] 上下文比较两个文件 -u 合并模式

diff [dir1] [dir2] 比较两个目录 -q 不需要比较同名文件内容

diff -u file1 file2 > file.patch 将两个文件不同找出并放到file.patch中去

patch file1 file.patch 对file1进行补丁修改

paste -d, file1 file2 以,为间隔 (默认是制表符为间隔) 合并file1和file2对应的行为一行。-s是串行处理和cat一样都不改变源文件, 需要使用 > [newfile] 输出到新文件

tr 工具是一个对一个字符替换

tr 'a-z' 'A-Z' < [file] 将file里的小写字母全部替换为大写字母 (不修改源文件) < 表示将文件内容作为标准输入进行文本编辑操作 0-9表示所有数字 -d表示删除指定字符 -s 压缩指定字符的连续重复

tr ':' '\t' > abc.txt == tr ':' '\t' | tee abc.txt

别名添加

vim /etc/bash.bashrc

在最后一行:

alias grep='grep --color=auto'

source /etc/bash.bashrc

root切换其他用户: su - [username]

head [file] > [targetfile] 将file前10行重定向到targetfile中

touch dir1/file{1..5} 创建file1到file5文件

ll -R dir* 查看dir开头的目录

:set list 看vim打开文件的控制字符

综合练习, 获取主机IP:

ifconfig eth0 | grep 'Bcast' | cut -d: -f2 | tr -d 'a-zA-z '

^c	终止前台运行的程序
^z	将前台运行的程序挂起到后台
^d	退出 等价exit
^l	清屏
^a home	光标移到命令行的最前端
^e end	光标移到命令行的后端
^u	删除光标前所有字符
^k	删除光标后所有字符
^r	搜索历史命令

{1..13} 表示1到13的序列

*: 匹配0或多个任意字符

?: 匹配任意单个字符

[list]: 匹配[list]中的任意单个字符, 或者一组单个字符 [a-z]

[!list]: 匹配除list中的任意单个字符

{string1,string2,...}: 匹配string1,string2或更多字符串]

rm -f file*

cp *.conf /dir1

touch file{1..5}

date +%F 日期

C++ Small Ticks

2021年10月28日 10:45

string转int的方式

采用最原始的string, 然后按照十进制的特点进行算术运算得到int, 但是这种方式太麻烦, 这里不介绍了。

采用标准库中atoi函数。

```
string s = "12";
int a = atoi(s.c_str());
对于其他类型也都有相应的标准库函数, 比如浮点型 atof(),long型 atol()等等。
```

采用sstream头文件中定义的字符串流对象来实现转换。

```
istringstream is("12"); //构造输入字符串流, 流的内容初始化为 “12”的字符串
int i;
is >> i; //从is流中读入一个int整数存入i中
```

二、int转string的方式

采用标准库中的to_string函数。

```
int i = 12;
cout << std::to_string(i) << endl;
不需要包含任何头文件, 应该是在utility中, 但无需包含, 直接使用, 还定义任何其他内置类型转为string的重载函数, 很方便。
```

采用sstream中定义的字符串流对象来实现。

```
ostringstream os; //构造一个输出字符串流, 流内容为空
int i = 12;
os << i; //向输出字符串流中输出int整数i的内容
cout << os.str() << endl; //利用字符串流的str函数获取流中的内容
字符串流对象的str函数对于istringstream和ostringstream都适用, 都可以获取流中的内容。
```

位运算 $\texttt{x \& -x}$ 取出 xx 的二进制表示中最低位那个 11, 设其为第 ll 位, 那么 x_1x_1

和 x_2x_2

中的某一个数的二进制表示的第 ll 位为 00, 另一个数的二进制表示的第 ll 位为 11。在这种情况下, $x_1 \oplus x_2x_1$

$\oplus x_2$

的二进制表示的第 ll 位才能为 11。

这样一来, 我们就可以把 \texttt{nums} 中的所有元素分成两类, 其中一类包含所有二进制表示的第 ll 位为 00 的数, 另一类包含所有二进制表示的第 ll 位为 11 的数。可以发现:

对于任意一个在数组 \texttt{nums} 中出现两次的元素, 该元素的两次出现会被包含在同一类中;

对于任意一个在数组 \texttt{nums} 中只出现了一次的元素, 即 x_1x_1

和 x_2x_2

, 它们会被包含在不同类中。

因此, 如果我们将每一类的元素全部异或起来, 那么其中一类会得到 x_1x_1

, 另一类会得到 x_2x_2

。这样我们就找出了这两个只出现一次的元素。

Markdown format

2021年10月29日 12:59

Markdown快速入门常用快捷键 (typora)
Markdown快速入门常用快捷键 (typora)
官方网站: <https://www.typora.io/>

支持平台:

Windows
Linux
OSX
特点:

完美支持Github的Markdown的语法;

人性化的书写方式:

表格的书写、挪动;
图片、超链接、网页表格复制;
目录生成;
支持LeTex公式书写;

支持Flowchart,Mermaid等流程图绘制;

emoji, 高亮, 备注, 上标, 下标等书写;

生成网页, PDF, 图片, 甚至word, LeTex等格式。

Markdown中常用的快捷键
Ctrl 0 到 Ctrl 6: 普通文本、一级文本~六级文本

Ctrl B: 加粗; 加粗测试

Ctrl I: 斜体; 斜体测试

Ctrl U: 下划线; 下划线测试

Shift Alt 5: 删除线; 删除线测试

Shift Ctrl ~: 行内代码块; 行内代码块测试

Ctrl K: 超链接, [超链接测试; 欢迎点一个大大的关注! ! !](《LL》-博客园(cnblogs.com)); 还支持文章内锚点, 按Ctrl 键点击此处 ③第一节

Ctrl T: 表格, 支持拖拽移动、网页端表格复制转换

Ctrl Shift Q: 引用;

Shift Ctrl I: 插入图片;

Shift Ctrl M: 公式块;

- []: 任务列表(可勾选的序列)注意每一个符号之间都有空格

^{内容}: 上标; 上标测试

_{内容}: 下标; 下标测试

:smile:: ☺

[toc]: 展示目录

Ctrl I: 选中一行

Ctrl d: 选中内容/单词

Ctrl home: 跳转到文章开头

Ctrl end: 跳转到文章结尾

Ctrl f: 搜索

Ctrl h: 替换

[^内容]: 脚注标识

Typora快捷键

无序列表: 输入-之后输入空格

有序列表: 输入数字+“.”之后输入空格

任务列表: -[空格]空格 文字

标题: ctrl+数字

表格: ctrl+t

生成目录: [TOC]按回车

选中一整行: ctrl+l

选中单词: ctrl+d

选中相同格式的文字: ctrl+e

跳转到文章开头: ctrl+home

跳转到文章结尾: ctrl+end

搜索: ctrl+f

替换: ctrl+h

引用: 输入>之后输入空格

代码块: ctrl+alt+f

加粗: ctrl+b

倾斜: ctrl+i

下划线: ctrl+u

删除线: alt+shift+5

插入图片: 直接拖动到指定位置即可或者ctrl+shift+i

插入链接: ctrl+k

Shift+Tab键 格式化代码

一、字体样式

斜体

一个*号或者使用快捷键Ctrl+I

我是斜体

粗体

两个*号或者使用快捷键Ctrl+B

我是粗体

粗斜

我是粗斜体

分割线

四个****号加回车

下划线

快捷键Ctrl+U</u>

删除线

两个波浪线

~~这个不常用~~

二、标题

一级标题

一个#号或者使用快捷键Ctrl+1

#我是一级标题#

二级标题

两个##号或者使用快捷键Ctrl+2

##我是二级标题##

依次类推, 到六级标题。正文使用Ctrl+0

三、列表

有序列表

数字 + . + space, 使用回车会自动生成下一个序号

数字.space

无序列表

*space或者是-space, 或者是+space

*space

四、引用和代码块

引用

英文状态下>space

鲁迅说: 世界上没来没有路。

代码块

英文状态下esc下面的那个键按三次```加回车, 选择语言。

快捷键Ctrl+Shift+Tab

五、插入图片和超链接

插入图片

![图片名](url)

快捷键Ctrl+Shift+I

也可以没有图片名，url也可以是本地文件路径，最好使用相对路径放在同一文件夹下。

插入超链接

[名字](url)

快捷键Ctrl+K

需要注意的是所有的符号都必须是英文状态下的。创建完成后点击就可以跳转了。

博客园

六、创建表格

快捷键Ctrl+T

| 姓名 | 年龄 | 出生地 | | --- | --- | ----- | | | | | | | | | | |

七、上标和下标

上标

X^{2}

X^{2}

下标

$H_{2}O$

$H_{2}O$

八、着重显示和高亮显示

着重显示

着重显示内容

高亮显示

==高亮显示==，不知道为什么这里不奏效，等后续再说吧。

我是尾巴

第一次使用markdown编辑文档，觉得还不错，相比于word能够更加专注内容。其实关于markdown还有很多东西要学，除了LaTex公式支持，海鸥流程图FlowChart，Mermaid，Sequence等时序图、甘特图等等。

最后推荐：

gif制作工具推荐

故不行硅步无以至千里，不积小流无以成江海！

RISC-V指令特点

2021年10月30日 15:29

- 1, R op R => R 从寄存器组里取两个数进行运行最后存入寄存器组。
- 2, R op IMM => R 从寄存器取一个数，与立即数IMM 进行操作后存入寄存器组。
- 3. 影响PC的指令，包含条件转移和无条件转移。如果指令顺序执行，下一条指令的指令地址是PC+4，而运行了这些影响PC的指令，就可能打破这个执行顺序（无条件跳转指令肯定影响PC）。因为无条件转移指令太少了只有两条（jal和jalr）所以不单独分类出来。
- 4, 读写存储器的指令。这里就有保存字（32位： word），半字(16位： half word)，字节(8位： byte)指令。以及加载字，半字，字节指令。其中我们可以将半字和字节当做有符号数以及无符号数据，加载半字就有了lh和lhu的区别，以及字节就有了lb和lbu的区别。另外在riscv64里面还有double word，两个word，每个字是32BIT，自然就是64位了。在64位指令系统还有了lw和lwu的区别。

官方将RV32I划分层了5种类型，这个分类是根据指令的机构进行分类的。

31	27	26	25	24	20	19	15	14	12	11	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type
imm[11:0]						rs1		funct3		rd		opcode		I-type
imm[11:5]				rs2		rs1		funct3		imm[4:0]		opcode		S-type
imm[12:10:5]				rs2		rs1		funct3		imm[4:1:11]		opcode		B-type
				imm[31:12]						rd		opcode		U-type
				imm[20:10:1][11:19:12]						rd		opcode		J-type

RISC-V 常见指令
算术运算

add rd, rs1, rs2

x[rd] = x[rs1] + x[rs2]

把寄存器 x[rs2]加到寄存器 x[rs1]上，结果写入 x[rd]。忽略算术溢出。

addi rd, rs1, immediate

x[rd] = x[rs1] + sext(immediate)

把符号位扩展的立即数加到寄存器 x[rs1]上，结果写入 x[rd]。忽略算术溢出。

sub rd, rs1, rs2

x[rd] = x[rs1] - x[rs2]

x[rs1]减去 x[rs2]，结果写入 x[rd]。忽略算术溢出。

div rd, rs1, rs2

x[rd] = x[rs1] ÷s x[rs2]

用寄存器 x[rs1]的值除以寄存器 x[rs2]的值，向零舍入，将这些数视为二进制的补码，把商写 入 x[rd]。

mul rd, rs1, rs2

x[rd] = x[rs1] × x[rs2]

把寄存器 x[rs2]乘到寄存器 x[rs1]上，乘积写入 x[rd]。忽略算术溢出。

rem rd, rs1, rs2

x[rd] = x[rs1] %s x[rs2]

求余数。x[rs1]除以 x[rs2]，向 0 舍入，都视为 2 的补码，余数写入 x[rd]。

neg rd, rs2

x[rd] = -x[rs2]

把寄存器 x[rs2]的二进制补码写入 x[rd]。

逻辑运算

and rd, rs1, rs2

RISC-V RV32标准指令集有以下几种框架：

- **R-format** for register-register arithmetic/logical operations
- **I-format** for register-immediate arith/logical operations and loads
- **S-format** for stores
- **B-format** for branches
- **U-format** for 20-bit upper immediate instructions
- **J-format** for jumps
- Others: Used for OS & Synchronization

R即Reg相关；**I**即立即数相关；**S**存储相关；**B**分支相关；**U**高位数相关（因为一条32位指令中无法表示高达32位的数据）；**J**跳转相关。

tips: about arithmetic & logical operations.

- 逻辑右移(LSR)是将各位依次右移指定位数，然后在**左侧补0**，算术右移(ASR)是将各位依次右移指定位数，然后在**左侧用原符号位补齐**。
- 逻辑左移与算术左移操作相同。
- RISC-V采用小端格式（Little-Endian），即低位字节排在内存的低地址端，高位字节排放在内存的高地址端。

R-Format

通常是xxx rd, rs1, rs2。

31
0

Funct7	rs2	rs1	Funct3	rd	opcode
7	5	5	3	5	7

Opcode = 0110011

Funct7 + Funct3 + opcode defines what operation to perform.

	Funct7	Funct3	opcode	用途
ADD	0000000	000	0110011	加法
SUB	0100000	000	0110011	减法
SLL	0000000	001	0110011	左移
SLT	0000000	010	0110011	Set Less := rs1 < rs2 ? 1:0
SLTU	0000000	011	0110011	SLT Unsigned
XOR	0000000	100	0110011	异或
SRL	0000000	101	0110011	逻辑右移
SRA	0100000	101	0110011	算术右移
OR	0000000	110	0110011	或
AND	0000000	111	0110011	与

相关伪指令：

mv rd, rs= addi rd, rs, x0

nop = addi r0, r0, x0

not rd, rs = xori rd, rs, 1111111111

Note: 因为某事突然发现RISC-V好像没有循环移位的指令（未查证），要实现循环移位估计要三条指令以上。

I-Format

Opcode 0010011

Immediate	rs1	Funct3	rd	opcode
-----------	-----	--------	----	--------

逻辑运算

and rd, rs1, rs2

$x[rd] = x[rs1] \& x[rs2]$

将寄存器 x[rs1]和寄存器 x[rs2]位与的结果写入 x[rd]。

andi rd, rs1, immediate

$x[rd] = x[rs1] \& sext(immediate)$

把符号位扩展的立即数和寄存器 x[rs1]上的值进行位与， 结果写入 x[rd]。

or rd, rs1, rs2

$x[rd] = \sim x[rs1]$

把寄存器 x[rs1]和寄存器 x[rs2]按位取或， 结果写入 x[rd]。

xor rd, rs1, immediate

$x[rd] = x[rs1] \wedge sext(immediate)$

x[rs1]和有符号扩展的 immediate 按位异或， 结果写入 x[rd]。

位运算

sll rd, rs1, rs2

$x[rd] = x[rs1] \ll x[rs2]$

逻辑左移（空位补0）

slli rd, rs1, shamt

立即数逻辑左移

srl rd, rs1, rs2

$x[rd] = (x[rs1] \gg u x[rs2])$

逻辑右移（空位补0）

srli rd, rs1, shamt

立即数逻辑右移

sra rd, rs1, rs2

$x[rd] = (x[rs1] \gg s x[rs2])$

算术右移（空位用最高位填充）

srai rd, rs1, shamt

立即数逻辑右移

not td, rs1

$x[rd] = \sim x[rs1]$

把寄存器 x[rs1]对于 1 的补码（即按位取反的值） 写入 x[rd]。实际被扩展为 xori rd, rs1, -1。

条件控制指令

beq rs1, rs2, offset

if (rs1 == rs2) pc += sext(offset)

若寄存器 x[rs1]和寄存器 x[rs2]的值相等， 把 pc 的值设为当前值加上符号位扩展的偏移 offset。

L-Format

Opcode 0010011

Immediate	rs1	Funct3	rd	opcode
12	5	3	5	7

	Shift-by-immediate	rs1	Funct3	rd	opcode
0x00000	5	5	3	5	7

第一部分包括ADDI, SLTI, SLTIU, XORI, ORI, ANDI（立即数有12位）

和SLLI, SRLI, SRAI（立即数仅有5位）。

第二部分包括load instruction， 格式同ADDI（12位的立即数）

Load Opcode 0000011

有LB(load byte)， LH(load halfword=2 bytes)， LW, LD, LBU(load unsigned byte), LHU(load unsigned halfword)

S-Format

Opcode 0100011

imm[11:5]	rs2	rs1	Funct	imm[4:0]	opcode
7	5	5	3	5	7

包括SB, SW, SH, SD

B-Format

Opcode 1100011

imm[12]	imm[10:5]	rs2	rs1	Funct	imm[4:1]	imm[11]	opcode
1	6	5	5	3	4	1	7

可以表示-4096~4094的范围

Note: The 12 immediate bits encode even 13-bit signed byte offsets (lowest bit of offset is always zero, so no need to store it).

包括BEQ, BNE, BLT(branch less than: if [rs1] <[rs2], then branch)， BGE(branch greater than or equal: if [rs1]>=[rs2], then branch)， BLTU, BGEU。

相关伪指令:

beqz x1, label = beq x1, x0, label

bnez x1, label = bne x1, x0, label

U-Format

imm[31:12]	rd	opcode
20	5	7

LUI – Load Upper Immediate lui rd, upper im(20-bit)

e.x. lui x10, 0x87654 # x10 = original value → 0x87654000

AUIPC – Add Upper Immediate to PC

e.x. auipc t0, 1 # t0 = original value → PC + 0x00001000

auipc t0, 0 # t0 = original value → PC

相关伪指令:

加载立即数

bge rs1, rs2, offset

if (rs1 ≥ s rs2) pc += sext(offset)

若寄存器 x[rs1]的值大于等于寄存器 x[rs2]的值（均视为二进制补码），把 pc 的值设为当前 值加上符号位扩展的偏移 offset。

blt rs1, rs2, offset

if (rs1 < s rs2) pc += sext(offset)

若寄存器 x[rs1]的值小于寄存器 x[rs2]的值（均视为二进制补码），把 pc 的值设为当前值加 上符号位扩展的偏移 offset。

bne rs1, rs2, offset

if (rs1 ≠ rs2) pc += sext(offset)

若寄存器 x[rs1]和寄存器 x[rs2]的值不相等，把 pc 的值设为当前值加上符号位扩展的偏移 offset。

跳转指令

j offset

pc += sext(offset)

把 pc 设置为当前值加上符号位扩展的 offset，等同于 jal x0, offset。

jal rd, offset

x[rd] = pc+4; pc += sext(offset)

把下一条指令的地址 (pc+4)，然后把 pc 设置为当前值加上符号位扩展的offset。rd 默认为 x1。

jr rs1

pc = x[rs1]

把 pc 设置为 x[rs1]，等同于 jalr x0, 0(rs1)。

jalr rd, offset(rs1)

t = pc+4; pc =(x[rs1]+sext(offset))&~1; x[rd]= t

把 pc 设置为 x[rs1] + sign-extend(offset)，把计算出的地址的最低有效位设为 0，并将原 pc+4 的值写入 f[rd]。rd 默认为 x1。

ret

pc = x[1]

从子过程返回。实际被扩展为 jalr x0, 0(x1)。

加载与存储指令

la rd, symbol

x[rd] = &symbol

将 symbol 的地址加载到 x[rd]中。

li rd, immediate

x[rd] = immediate

将常量加载到 x[rd]中。

lw rd, offset(rs1)

x[rd] = sext(M[x[rs1] + sext(offset)][31:0])

从地址 x[rs1] + sign-extend(offset)读取四个字节，写入 x[rd]。

storew rd, offset(rs1)

li rd, imm(32-bit) = lui rd, imm(20-bit)

addi rd, rd, imm(12-bit)

加载地址

la rd, label= auipc rd, imm(20-bit)

addi rd, rd, imm(12-bit)

J-Format

imm[20]	imm[10: 1]	imm[11]	imm[19: 12]	rd	opcode
1	10	1	8	5	7

jal x0, label Discard return address

jal ra, function_name Call Function within 2¹⁸ (1 instruction = 2² bytes).

jr ra

Call function at any 32-bit absolute address

lui x1, <high 20-bit>

jalr ra, x1, <low 12-bit> Call Function at 32-bit absolute address

Jump PC-relative with 32-bit offset

auipc x1, <high 20-bit>

jalr x0, x1, <low 12-bit> Jump within 32-bit

相关伪指令:

j label = jal x0, label

ret = jr ra = jalr x0, ra, 0

Acknowledgement

- UC Berkeley CS61C
- <http://www.kvakil.me/venus/> 可以在线编译RISC-V的工具

来自 <<https://www.cnblogs.com/George-Okelly1995/p/9801357.html>>

sw rs2, offset(rs1)

$M[x[rs1] + sext(offset)] = x[rs2][31:0]$

将 x[rs2] 的低位 4 个字节存入内存地址 $x[rs1] + sign_extend(offset)$ 。

参考：

RISC-V 手册（翻译版） <http://crva.io/documents/RISC-V-Reader-Chinese-v2p1.pdf>

2021年11月2日 11:31

```
build/RISCV_iw16/gem5.opt --stats-file iw16_bzip_stats.txt ./configs/example/se.py -c ./spec/bin/bzip2 -
o "/spec/401.bzip2/chicken.jpg 30" -I=500000000 -F=100000000 --cpu-type O3CPU --caches
```

或者直接用(崔博:)

```
-I 500000000 -F 100000000
```

```
--checkpoint-restore=4 --restore-with-cpu=detailed --
checkpoint-dir
/home/sakshi/workspace2/gem5-63325e5b0a9d-
modified/m5out/libq_cpt/
参考: https://www.mail-archive.com/gem5-
users@gem5.org/msg15857.html
```

```

parser.add_argument(
    "--repeat-switch", action="store", type=int, default=None,
    help="Repeat each test and forth between CPU and GPU with period <val>")
parser.add_argument(
    "--", "--standard-switch", action="store", type=int, default=None,
    help="Switch from using the Detailed CPU after warmup period of <val>")
parser.add_argument(
    "--", "--warmup-interval", action="store", type=int, default=None,
    help="CPU Warmup Interval")

# Performance and checkpoint related materials
parser.add_argument(
    "--", "--save-ints", action="store", type=int, default=None,
    help="Save the ints and total instructions (requires --standard-switch)")
parser.add_argument(
    "--checkpoint", action="store", type=str, default=None,
    help="Path for --take-checkpoint and --checkpoint-restore")
parser.add_argument(
    "--", "--last-forward", action="store", type=str, default=None,
    help="Number of instructions to fast forward before switching")
parser.add_argument(
    "--", "--alignint", action="store_true", default=False,
    help="--alignint aligns as an instruction offset for --checkpoint-restore or --take-checkpoint")
parser.add_argument(
    "--", "--not-instruction", action="store_true", default=False,
    help="--not-instruction as --checkpoint-restore or --take-checkpoint as a number of instructions")
parser.add_argument(
    "--", "--save-logs", action="store", default=None,

```

liz:
备选项的话就 h24ref mcf sjeng

403.gcc.C语言实现，同spec2000的176.gcc类似。该测试是基于gcc V3.2,为AMDCPU生成机器码。相比spec2000的176.gcc,该测试有更多的输入文件，因此测试压力会更大，负载来源于对9组C代码进行编译。

429.mcf.C语言实现，同时需要libm库支持，同spec2000的181.mcf类似，MCF是一个用于大型公共交通中的单车辆调度的程序。但对输入文件做了一定的修改，由32位变成64位，用以兼容64位系统。并且增加了cache命中和程序的性能。相比spec2000的181.mcf来说，占用的内存由之前的100M-190M变为860M-1700M。

上面运行了一个程序，cpu数量为1。如果需要运行两个程序A，B，则将-n 1改成-n 2，同时修改-c、-o、等信息，具体为将A和B的-c放在一块，以“；”隔开，其他-o、--output、--errout、--input类

403.gcc: C语言实现, 同spec2000的170.gcc类似。该测试是于gcc v3.2.2/3.4.6/4.0上成功编译。相比spec2000的170.gcc, 该测试有更多的输入文件, 因此测试压力会更大, 负载来源于对9组C代码进行编译。

429.mcf: C语言实现, 同时需要libm库支持, 同spec2000的181.mcf类似, MCF是一个用于大型公共交通中的单站车辆调度的程序。但对输入文件做了一定的修改, 由32位变成64位, 用以兼容64位系统。并且增加了cache命中和程序的性能。相比spec2000的181.mcf来说, 占用的内存由之前的100M-190M变为860M-1700M。

445.gobmk: C语言实现, 同spec2000的186.crafty类似。不同的是这里是实现的围棋游戏。相比spec2000程序更复杂。

456.hmmer: C语言实现。HMMER是基于隐马尔可夫模型(profile HMMs), 用于生物序列分析工作。同Timed HMMer Search 类似。

458.sjeng: C语言实现。基于一种象棋游戏Sjeng11.2, 属于人工智能的范畴。

462.libquantum: C语言实现 (C99)。libquantum是模拟量子计算机的库文件, 用来进行量子计算机应用的研究。

464.h264ref: C语言实现。一种视频压缩程序, 基于H264AVC 9.3版, 去除了I/O和平台相关的东西。

471.omnetpp: C++语言实现。OMNeT++, 离散事件仿真。包括约8000台计算机和900个交换机/集线器, 以及混合了各种从10Mb到1000Mb速率的大型CSMA/CD协议以太网网络模拟。

473.astar: C++语言实现, 实现了2D寻路算法A*的三种不同版本。

481.wrf: Fortran 90 & C语言实现。WRF v2.0.2 481.wrf基于WRF(Weather Research and Forecastin)模型, 对NCAR的数据进行了计算, 数据包括了UTC 2001.06.11到UTC 2001.06.12以三小时为间隔的数据

482.sphinx3: C语言实现。Sphinx-3一种语音识别软件。

上面运行了一个程序, cpu数量为1。如果需要运行两个程序A, B, 则将-n 1改成-n 2, 同时修改-c、-o、等信息, 具体为将A和B的-c放在一块, 以“;”隔开, 其他-o、--output、--errout、--input类似, 如果某一个程序选项为空, 也需要隔开, 保证二进制文件与选项、输出、输入的对对应关系。

一个2个程序的例子如下, 只显示重要部分:

```
-c
"/home/feng/spec2006-12.04/benchspec/CPU2006/401.bzip2/run/run_base_ref_amd64-m64-gcc43-nn.0000/bzip2_base.amd64-m64-gcc43-nn; /home/feng/spec2006-12.04/benchspec/CPU2006/403.gcc/run/run_base_ref_amd64-m64-gcc43-nn.0000/gcc_base.amd64-m64-gcc43-nn"

-o
"/home/feng/spec2006-12.04/benchspec/CPU2006/401.bzip2/run/run_base_ref_amd64-m64-gcc43-nn.0000/input.source.280; /home/feng/spec2006-12.04/benchspec/CPU2006/403.gcc/run/run_base_ref_amd64-m64-gcc43-nn.0000/166.io.166.s"

--output="input.source.out; 166.out"

--errout="input.source.err; 166.err"
```

为了方便, 可以在GEM5目录下, 创建文件shellRunMultipleSpecsh, 将命令写入该文件, 之后运行#shellRunMultipleSpecsh, 即可运行shellRunMultipleSpecsh中的命令。

在运行spec2006的某些程序时, 可能会出现错误, 一般是路径的问题, 即程序找不到某个文件, 可以将文件从SPEC 2006中直接复制到GEM5目录下。当出现错误时, 可以查看对应输出文件中的错误类型, 进行处理。

Network

2021年11月4日 16:02

Win10打不开OneDrive提示“连接到OneDrive时出现问题”怎么解决？很多伙伴在操作Win10系统的时候，都会遇到OneDrive打不开的情况，OneDrive会显示灰色状态，如果双击打开的话，系统会提示“连接到OneDrive时出现问题”，这该怎么办呢？一方面我们可以尝试重置网络，另一方面可以重置OneDrive服务，下面请看具体的操作方法。

注：出现这样的问题首先确保当前网络正常，网页正常打开，正常登录QQ，正常打开商店等等，然后尝试重置Win10中的OneDrive。

首先尝试重置网络：

- 1、在开始菜单单击鼠标右键，会弹出一个菜单，选择“命令提示符（管理员）”；
- 2、输入：netsh int ip reset c:\resetlog.txt，按下回车键；
- 3、再输入netsh winsock reset命令后回车，然后重启计算机查看能否解决。

OneDrive 重置方法如下：

- 1、键盘上按下Windows键 + R 组合键打开运行，在弹出的框中输入：%localappdata%\Microsoft\OneDrive\onedrive.exe /reset然后点击【确定】（桌面右下角的OneDrive系统小图标将消失并在大约1-2分钟后再次出现）；
- 2、重置后若在任务栏右下角的在数分钟后并没有再次出现OneDrive图标的话再次按下win+R 打开运行，输入：%localappdata%\Microsoft\OneDrive\onedrive.exe 按下回车键即可！
- 3、重置完成后，请右击OneDrive小图标并选择 设置 /Settings，选择希望同步的文件夹确保希望被同步的文件夹已被打勾选取即可！

以上便是Win10打不开OneDrive提示“连接到OneDrive时出现问题”的解决办法，出现这样的问题大多是由网络问题导致的，大家可以尝试重置网络来解决，如果网络没问题，那么就重置OneDrive，相信你的问题会得到解决的。

来自 <<https://zhidao.baidu.com/question/1386042727522539460.html>>

Linux常用指令

2021年11月7日 17:19

find 搜索路径 [选项] 搜索内容

选项:

- -name: 按照文件名搜索;
- -iname: 按照文件名搜索, 不区分文件大小;
- -inum: 按照 inode 号搜索;

这是 find 最常用的用法, 我们来试试:

```
[root@localhost ~]# find /-name yum.conf
/etc/yum.conf
```

#在目录下查找文件名是yum.conf的文件

但是 find 命令有一个小特性, 就是搜索的文件名必须和你的搜索内容一致才能找到。如果只包含搜索内容, 则不会找到。我们做一个实验:

```
[root@localhost ~]# touch yum.conf.bak
```

#在/root/目录下建立一个文件yum.conf.bak

```
[root@localhost ~]# find /-name yum.conf
/etc/yum.conf
```

#搜索只能找到yum.conf文件, 而不能找到 yum.conf.bak 文件

find 能够找到的是只有和搜索内容 yum.conf 一致的 /etc/yum.conf 文件, 而 /root/yum.conf.bak 文件虽然含有搜索关键字, 但是不会被找到。这种特性我们总结为: **find 命令是完全匹配的, 必须和搜索关键字一模一样才会列出。**

Linux 中的文件名是区分大小写的, 也就是说, 搜索小写文件, 是找不到大写文件的。如果想要大小通吃, 就要使用 -iname 来搜索文件。

```
[root@localhost ~]# touch CANGLS
```

```
[root@localhost ~]# touch cangls
```

#建立大写和小写文件

```
[root@localhost ~]# find.-iname cangls
```

```
./CANGLS
```

```
./cangls
```

#使用-iname, 大小写文件通吃

每个文件都有 inode 号, 如果我们知道 inode 号, 则也可以按照 inode 号来搜索文件。

```
[root@localhost ~]# ls -li install.log
```

```
262147 install.log
```

#如果知道文件名, 则可以用"ls -li"来查找inode号

```
[root@localhost ~]# find.-inum 262147
```

```
./install.log
```

#如果知道inode号, 则可以用find命令来查找文件

按照 inode 号搜索文件, 也是区分硬链接文件的重要手段, 因为硬链接文件的 inode 号是一致的。

```
[root@localhost ~]# ln /root/install.log /tmp/
```

#给install.log文件创建一个硬链接文件

```
[root@localhost ~]# ll -li /root/install.log /tmp/install.log
```

```
262147 -rw-r--r--.2 root root 24772 1月 14 2014/root/
install.log
```

```
262147 -rw-r--r--.2 root root 24772 1月 14 2014/tmp/
install.log
```

#可以看到这两个硬链接文件的inode号是一致的

```
[root@localhost ~]# find /-inum 262147
```

```
/root/install.log
```

```
/tmp/install.log
```

#如果硬链接不是我们自己建立的, 则可以通过find命令搜索inode号, 来确定硬链接文件

按照文件大小搜索

```
[root@localhost ~]# find 搜索路径 [选项] 搜索内容
```

选项:

• -size: 按照文件大小搜索;

ctrl + r (to find history)

The Best Keyboard Shortcuts for Bash (aka the Linux and macOS Terminal)



LOWELL HEDDINGS

MAR 17, 2017, 6:40 AM EST | 4 MIN READ

```
chrish@ubuntu:~$ bash --version
GNU bash, version 4.3.46(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.
This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
chrish@ubuntu:~$
```

Bash is the default [command-line shell](#) on most Linux distributions, from Ubuntu and Debian to Red Hat and Fedora. Bash is also the default shell included with macOS, and you can [install a Linux-based bash environment on Windows 10](#).

The bash shell features a wide variety of keyboard shortcuts you can use. These will work in bash on any operating system. Some of them may not work if you're accessing bash remotely through an SSH or telnet session, depending on how you have your keys mapped.

RELATED: [10 Basic Linux Commands for Beginners](#)

Working With Processes

Use the following shortcuts to manage running processes.

- Ctrl+C: Interrupt (kill) the current foreground process running in the terminal. This sends the [SIGINT](#) signal to the process, which is technically just a request—most processes will honor it, but some may ignore it.
- Ctrl+Z: Suspend the current foreground process running in bash. This sends the SIGTSTP signal to the process. To return the process to the foreground later, use the fg process_name command.
- Ctrl+D: Close the bash shell. This sends an EOF (End-of-file) marker to bash, and bash exits when it receives this marker. This is similar to running the exit command.

```
chrish@ubuntu:~$ ping google.com
PING google.com (216.58.216.142) 56(84) bytes of data:
64 bytes from 216.58.216.142: icmp_seq=1 ttl=35.8 ms
```

```
[root@localhost ~]#find 搜索路径 [选项] 搜索内容
```

选项:

- -size[+-]大小: 按照指定大小搜索文件

这里的"+"的意思是搜索比指定大小还要大的文件, "-"的意思是搜索比指定大小还要小的文件。我们来试试:

```
[root@localhost ~]# ll -h install.log
```

```
-rw-r--r--.1 root root 25K 1月 14 2014 install.log #在当前目录下有一个大小是25KB的文件
```

```
[root@localhost ~]#
```

```
[root@localhost ~]# find.-size 25k
```

```
./install.log
```

#当前目录下, 查找大小刚好是25KB的文件, 可以找到

```
[root@localhost ~]# find.-size -25k
```

```
.
```

```
./bashrc
```

```
./viminfo
```

```
./tcshrc
```

```
./pearcc
```

```
./anaconda-ks.cfg
```

```
./test2
```

```
./ssh
```

```
./bash_history
```

```
./lessht
```

```
./bash_profile
```

```
./yum.conf.bak
```

```
./bashjogout
```

```
./install.log.syslog
```

```
./cshrc
```

```
./cangls
```

#搜索小于25KB的文件, 可以找到很多文件

```
[root@localhost ~]# find.-size +25k
```

#而当前目录下没有大于25KB的文件

其实 find 命令的 -size 选项是笔者个人觉得比较恶心的选项, 为什么这样说? find 命令可以按照 KB 来搜索, 应该也可以按照 MB 来搜索吧。

```
[root@localhost ~]# find.-size -25m
```

```
find:无效的-size类型"m"
```

#为什么会报错呢? 其实是因为如果按照MB来搜索, 则必须是大写的M

这就是纠结点, 千字节必须是小写的"k", 而兆字节必须是大写的"M"。有些人会说: "你别那么执着啊, 你就不能不写单位, 直接按照字节搜索啊? "很傻, 很天真, 不写单位, 你们就以为会按照字节搜索吗? 我们来试试:

```
[root@localhost ~]# ll anaconda-ks.cfg
```

```
-rw-----.1 root root 1207 1月 14 2014 anaconda-ks.cfg
```

#anaconda-ks.cfg文件有1207字节

```
[root@localhost ~]# find.-size 1207
```

#但用find查找1207, 是什么也找不到的

也就是说, find 命令的默认单位不是字节。如果不写单位, 那么 find 命令是按照 512 Byte 来进行查找的。我们看看 find 命令的帮助。

```
[root@localhost ~]# man find
```

```
-size n[cwbkMG]
```

File uses n units of space. The following suffixes can be used:

'b' for 512-byte blocks (this is the default if no suffix is used)

#这是默认单位, 如果单位为b或不写单位, 则按照 512Byte搜索

'c' for bytes

#搜索单位是c, 按照字节搜索

'w' for two-byte words

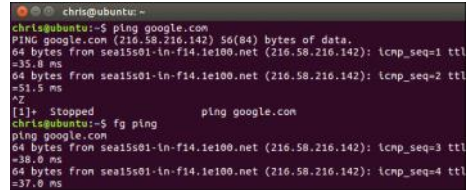
#搜索单位是w, 按照双字节 (中文) 搜索

'k' for Kilobytes (units of 1024 bytes)

#按照KB单位搜索, 必须是小写的k

'M' for Megabytes (units of 1048576 bytes)

the exit command.



RELATED: [How Linux Signals Work: SIGINT, SIGTERM, and SIGKILL](#)

Controlling the Screen

The following shortcuts allow you to control what appears on the screen.

- Ctrl+L: Clear the screen. This is similar to running the "clear" command.

- Ctrl+S: Stop all output to the screen. This is particularly useful when running commands with a lot of long, verbose output, but you don't want to stop the command itself with Ctrl+C.

- Ctrl+Q: Resume output to the screen after stopping it with Ctrl+S.

Moving the Cursor

Use the following shortcuts to quickly move the cursor around the current line while typing a command.

- Ctrl+A or Home: Go to the beginning of the line.

- Ctrl+E or End: Go to the end of the line.

- Alt+B: Go left (back) one word.

- Ctrl+B: Go left (back) one character.

- Alt+F: Go right (forward) one word.

- Ctrl+F: Go right (forward) one character.

- Ctrl+XX: Move between the beginning of the line and the current position of the cursor. This allows you to press Ctrl+XX to return to the start of the line, change something, and then press Ctrl+XX to go back to your original cursor position. To use this shortcut, hold the Ctrl key and tap the X key twice.

Deleting Text

Use the following shortcuts to quickly delete characters:

- Ctrl+D or Delete: Delete the character under the cursor.

- Alt+D: Delete all characters after the cursor on the current line.

- Ctrl+H or Backspace: Delete the character before the cursor.

Fixing Typos

These shortcuts allow you to fix typos and undo your key presses.

- Alt+T: Swap the current word with the previous word.

- Ctrl+T: Swap the last two characters before the cursor with each other. You can use this to quickly fix typos when you type two characters in the wrong order.

- Ctrl+_ : Undo your last key press. You can repeat this to undo multiple times.

M for Megabytes (units of 1048576 bytes)

#按照MB单位搜索，必须是大写的M

'G' for Gigabytes (units of 1073741824 bytes)

#按照GB单位搜索，必须是大写的G

也就是说，如果想要按照字节搜索，则需要加搜索单位“c”。我们来试试：

```
[root@localhost ~]# find.-size 1207c
```

```
./anaconda-ks.cfg
```

#使用搜索单位c，才会按照字节搜索

按照修改时间搜索

Linux 中的文件有访问时间(atime)、数据修改时间(mtime)、状态修改时间(ctime)这三个时间，我们也可以按照时间来搜索文件。

```
[root@localhost ~]# find搜索路径 [选项] 搜索内容
```

选项：

- atime [+ -]时间: 按照文件访问时间搜索
- mtime [+ -]时间: 按照文件修改时间搜索
- ctime [+ -]时间: 按照文件修改时间搜索

这三个时间的区别我们在 stat 命令中已经解释过了，这里用 mtime 数据修改时间来举例，重点说说 “[+ -]” 时间的含义。

- 5: 代表@内修改的文件。
- 5: 代表前5~6天那一天修改的文件。
- +5: 代表6天前修改的文件。

我们画一个时间轴，来解释一下，如图 1 所示。

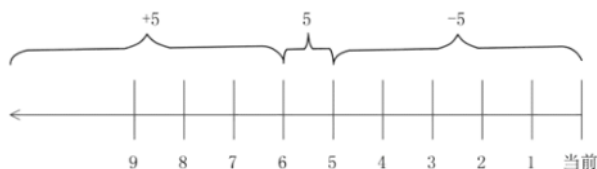


图 1 find时间轴

每次笔者讲到这里，“-5”代表 5 天内修改的文件，而“+5”总有人说代表 5 天修改的文件。要是笔者能知道 5 天系统中能建立什么文件，早就去买彩票了，那是未卜先知啊！所以“-5”指的是 5 天内修改的文件，“5”指的是前 5~6 天那一天修改的文件，“+5”指的是 6 天前修改的文件。我们来试试：

```
[root@localhost ~]# find.-mtime -5
```

#查找5天内修改的文件

大家可以在系统中把几个选项都试试，就可以明白各选项之间的差别了。

find 不仅可以按照 atime、mtime、ctime 来查找文件的时间，也可以按照 amin、mmin 和 cmin 来查找文件的时间，区别只是所有 time 选项的默认单位是天，而 min 选项的默认单位是分钟。

按照权限搜索

在 find 中，也可以按照文件的权限来进行搜索。权限也支持 [+/-] 选项。我们先看一下命令格式。

```
[root@localhost ~]# find 搜索路径 [选项] 搜索内容
```

选项：

- perm 权限模式: 查找文件权限刚好等于“权限模式”的文件
- perm -权限模式: 查找文件权限全部包含“权限模式”的文件
- perm +权限模式: 查找文件权限包含“权限模式”的任意一个权限的文件

为了便于理解，我们要举几个例子。先建立几个测试文件。

```
[root@localhost ~]# mkdir test
```

```
[root@localhost ~]# cd test/
```

```
[root@localhost test]# touch test1
```

```
[root@localhost test]# touch test2
```

```
[root@localhost test]# touch test3
```

```
[root@localhost test]# touch test4
```

#建立测试目录，以及测试文件

```
[root@localhost test]# chmod 755 test1
```

```
[root@localhost test]# chmod 444 test2
```

```
[root@localhost test]# chmod 600 test3
```

```
[root@localhost test]# chmod 200 test4
```

#设定实验权限。因为是实验权限，所以看起来比较别扭

Cutting and Pasting

Bash includes some basic cut-and-paste features.

- Ctrl+W: Cut the word before the cursor, adding it to the clipboard.
- Ctrl+K: Cut the part of the line after the cursor, adding it to the clipboard.
- Ctrl+U: Cut the part of the line before the cursor, adding it to the clipboard.
- Ctrl+Y: Paste the last thing you cut from the clipboard. The y here stands for “yank” .

Capitalizing Characters

The bash shell can quickly convert characters to upper or lower case:

THE BEST TECH NEWSLETTER ANYWHERE

Join **425,000** subscribers and get a daily digest of features, articles, news, and trivia.

Sign Me Up!

By submitting your email, you agree to the [Terms of Use](#) and [Privacy Policy](#).

- Alt+U: Capitalize every character from the cursor to the end of the current word, converting the characters to upper case.
- Alt+L: Uncapitalize every character from the cursor to the end of the current word, converting the characters to lower case.
- Alt+C: Capitalize the character under the cursor. Your cursor will move to the end of the current word.

Tab Completion

RELATED: [Use Tab Completion to Type Commands Faster on Any Operating System](#)

Tab completion is a very useful bash feature. While typing a file, directory, or command name, press Tab and bash will automatically complete what you’re typing, if possible. If not, bash will show you various possible matches and you can continue typing and pressing Tab to finish typing.

- Tab: Automatically complete the file, directory, or command you’re typing.

For example, if you have a file named really_long_file_name in /home/chris/ and it’s the only file name starting with “r” in that directory, you can type /home/chris/r, press Tab, and bash will automatically fill in /home/chris/really_long_file_name for you. If you have multiple files or directories starting with “r”, bash will inform you of your possibilities. You can start typing one of them and press “Tab” to continue.

```
chris@ubuntu:~$ cp /home/chris/r
random_file really_long_file_name
chris@ubuntu:~$ cp /home/chris/re
```

Working With Your

Command History

RELATED: [How to Use Your Bash History in the Linux or macOS Terminal](#)

You can quickly scroll through your recent commands, which are stored in your user

#设定实验权限。因为是实验权限，所以看起来比较别扭

```
[root@localhost test]# ll
```

总用量0

```
-rwxr-xr-x 1 root root 0 6月 17 11:05 test1 -r--r--r-- 1 root root 0 6月 17 11:05 test2
```

```
-rw----- 1 root root 0 6月 17 11:05 test3
```

```
-w----- 1 root root 0 6月 17 11:05 test4
```

#查看权限

【例 1】"-perm权限模式"。

这种搜索比较简单，代表查找的权限必须和指定的权限模式一模一样，才可以找到。

```
[root@localhost test]#find.-perm 444
```

```
./test2
```

```
[root@localhost test]#find.-perm 200
```

```
./test4
```

#按照指定权限搜索文件，文件的权限必须和搜索指定的权限一致，才能找到

【例 2】"-perm-权限模式"。

如果使用"-权限模式"，是代表的是文件的权限必须全部包含搜索命令指定的权限模式，才可以找到。

```
[root@localhost test]#find.-perm -200
```

```
./test4 <-此文件权限为200
```

```
./test3 <-此文件权限为600
```

```
./test1 <-此文件权限为755
```

#搜索文件的权限包含200的文件，不会找到test2文件，因为test2的权限为444，不包含200权限

因为 test4 的权限 200(-w-----)、test3 的权限 600(-rw-----)和 test1 的权限 755(-rwxr-xr-x) 都包含 200(-w-----) 权限，所以可以找到；而 test2 的权限是 444 (-r--r--r--)，不包含 200 (-w-----)权限，所以找不到，再试试：

```
[root@localhost test]# find.-perm -444
```

```
.
```

```
./test2 <-此文件权限为444
```

```
./test1 <-此文件权限为755
```

#搜索文件的权限包含444的文件

上述搜索会找到 test1 和 test2，因为 test1 的权限 755 (-rwxr-xr-x)和 test2 的权限 444 (-r--r--r--)都完全包含 444 (-r--r--r--)权限，所以可以找到；而 test3 的权限 600 (-rw-----)和 test4 的权限 200 (-w-----)不完全包含 444 (-r--r--r--) 权限，所以找不到。也就是说，test3 和 test4 文件的所有者权限虽然包含 4 权限，但是所属组权限和其他人权限都是 0，不包含 4 权限，所以找不到，这也是完全包含的意义。

【例 3】"-perm+权限模式"

刚刚的"-perm-权限模式"是必须完全包含，才能找到；而"-perm+权限模式"是只要包含任意一个指定权限，就可以找到。我们来试试：

```
[root@localhost test]# find.-perm +444
```

```
./test4 <-此文件权限为200
```

```
./test3 <-此文件权限为600
```

```
./test1 <-此文件权限为755
```

#搜索文件的权限包含200的文件，不会找到test2文件，因为test2的权限为444，不包含200权限。

因为 test4 的权限 200 (-w-----)、test3 的权限 600 (-rw-----)和 test1 的权限 755 (-rwxr-xr-x)都包含 200(-w-----)权限，所以可以找到；而 test2 的权限是 444 (-r--r--r--)，不包含 200 (-w-----)权限，所以找不到。

按照所有者和所属组搜索

```
[root@localhost ~]# find 搜索路径 [选项] 搜索内容
```

选项：

- uid 用户 ID:按照用户 ID 查找所有者是指定 ID 的文件
- gid 组 ID:按照用户组 ID 查找所属组是指定 ID 的文件
- user 用户名: 按照用户名查找所有者是指定用户的文件
- group 组名: 按照组名查找所属组是指定用户组的文件
- nouser: 查找没有所有者的文件

这组选项比较简单，就是按照文件的所有者和所属组来进行文件的查找。在 Linux 系统中，绝大多数文件都是使用 root 用户身份建立的，所以在默认情况下，绝大多数系统文件的所有者都是 root。例如：

```
[root@localhost ~]#find.-user root
```

#在当前目录中查找所有者是 root 的文件

由于当前目录是 root 的家目录，所有文件的所有者都是 root 用户，所以这条搜索命令会找到当前目录下所有

commands, which are stored in your user account' s [bash history](#) file:

- Ctrl+P or Up Arrow: Go to the previous command in the command history. Press the shortcut multiple times to walk back through the history.

- Ctrl+N or Down Arrow: Go to the next command in the command history. Press the shortcut multiple times to walk forward through the history.

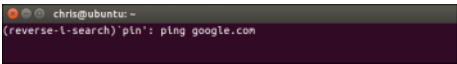
- Alt+R: Revert any changes to a command you' ve pulled from your history if you' ve edited it.

Bash also has a special "recall" mode you can use to search for commands you' ve previously run:

- Ctrl+R: Recall the last command matching the characters you provide. Press this shortcut and start typing to search your bash history for a command.

- Ctrl+O: Run a command you found with Ctrl+R.

- Ctrl+G: Leave history searching mode without running a command.



emacs vs. vi Keyboard

Shortcuts

The above instructions assume you' re using the default keyboard shortcut configuration in bash. By default, bash uses emacs-style keys. If you' re more used to the vi text editor, you can switch to [vi-style keyboard shortcuts](#).

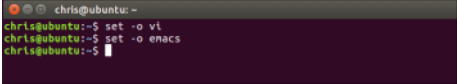
The following command will put bash into vi mode:

```
set -o vi
```

ADVERTISEMENT

The following command will put bash back into the default emacs mode:

```
set -o emacs
```



With a few of these in your toolbox, you' ll be a Terminal master in no time.

来自 <<https://www.howtogeek.com/howto/ubuntu/keyboard-shortcuts-for-bash-command-shell-for-ubuntu-debian-suse-redhat-linux-etc/>>

的文件。

按照所有者和所属组搜索时, "-nouser"选项比较常用, 主要用于查找垃圾文件。在 Linux 中, 所有的文件都有所有者, 只有一种情况例外, 那就是外来文件。比如光盘和 U 盘中的文件如果是由 Windows 复制的, 在 Linux 中查看就是没有所有者的文件; 再比如手工源码包安装的文件, 也有可能没有所有者。

除这种外来文件外, 如果系统中发现了没有所有者的文件, 一般是没有作用的垃圾文件(比如用户删除之后遗留的文件), 这时需要用户手工处理。搜索没有所有者的文件, 可以执行以下命令:

```
[root@localhost ~]# find/-nouser
```

按照文件类型搜索

```
[root@localhost ~]# find 搜索路径 [选项] 搜索内容
```

选项:

- -type d: 查找目录
- -type f: 查找普通文件
- -type l: 查找软链接文件

这个命令也很简单, 主要按照文件类型进行搜索。在一些特殊情况下, 比如需要把普通文件和目录文件区分开, 比如需要把普通文件和目录文件区分开, 使用这个选项就很方便。

```
[root@localhost ~]# find /etc -type d
```

#查找/etc/目录下有哪些子目录

逻辑运算符

```
[root@localhost ~]#find 搜索路径 [选项] 搜索内容
```

选项:

- -a: and逻辑与
- -o: or逻辑或
- -not: not逻辑非

1) -a:and逻辑与

find 命令也支持逻辑运算符选项, 其中 -a 代表逻辑与运算, 也就是 -a 的两个条件都成立, find 搜索的结果才成立。

举个例子:

```
[root@localhost ~]# find.-size +2k -a -type f
```

#在当前目录下搜索大于2KB, 并且文件类型是普通文件的文件

在这个例子中, 文件既要大于 2KB, 又必须是普通文件, find 命令才可以找到。再举个例子:

```
[root@localhost ~]# find.-mtime -3 -a -perm 644
```

#在当前目录下搜索3天以内修改过, 并且权限是644的文件

2) -o:or逻辑或

-o 选项代表逻辑或运算, 也就是 -o 的两个条件只要其中一个成立, find 命令就可以找到结果。例如:

```
[root@localhost ~]#find.-name cangls -o -name bols
```

```
./cangls
```

```
./bols
```

#在当前目录下搜索文件名要么是cangls的文件, 要么是bols的文件

-o 选项的两个条件只要成立一个, find 命令就可以找到结果, 所以这个命令既可以找到 cangls 文件, 也可以找到 bols 文件。

3) -not:not逻辑非

-not是逻辑非, 也就是取反的意思。举个例子:

```
[root@localhost ~]# find.-not -name cangls
```

#在当前目录下搜索文件名不是cangls的文件

其他选项

1) -exec选项

这里我们主要讲解两个选项"-exec"和"-ok", 这两个选项的基本作用非常相似。我们先来看看 "exec"选项的格式。

```
[root@localhost ~]# find 搜索路径 [选项] 搜索内容 -exec 命令2{\;
```

首先, 请大家注意这里的"{"和"}"是标准格式, 只要执行"-exec"选项, 这两个符号必须完整输入。

其次, 这个选项的作用其实是把 find 命令的结果交给由"-exec"调用的命令 2 来处理。"{"就代表 find 命令的查找结果。

我们举个例子，刚刚在讲权限的时候，使用权限模式搜索只能看到文件名，例如：

```
[root@localhost test]# find.-perm 444
./test2
```

如果要看文件的具体权限，还要用"ll"命令查看。用"-exec"选项则可以一条命令搞定：

```
[root@localhost test]# find.-perm 444 -exec ls -l {} \;
-r--r--r-- 1 root root 0 6月 17 11:05 ./test2
```

#使用"-exec"选项，把find命令的结果直接交给"ls -l"命令处理

"-exec"选项的作用是把 find 命令的结果放入"{}"中，再由命令 2 直接处理。在这个例子中就是用"ls -l"命令直接处理，会使 find 命令更加方便。

2) -ok选项

"-ok"选项和"-exec"选项的作用基本一致，区别在于："-exec"的命令会直接处理，而不询问；"-ok"的命令 2 在处理前会先询问用户是否这样处理，在得到确认命令后，才会执行。例如：

```
[root@localhost test]# find.-perm 444 -ok rm -rf {} \;
<rm....../test2> ?y <-需要用户输入y,才会执行
```

#我们这次使用rm命令来删除find找到的结果，删除的动作最好确认一下

来自 <<http://c.biancheng.net/view/779.html>>

zsh

2021年11月9日 0:44

查看bin下是否有zsh包，所有的shell列表

```
cat /etc/shells
```

切换到bin下面的zsh

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

按提示所述，shell已经更改为zsh了，现在查看一下系统当前使用的shell;

```
echo $SHELL
```

来自 <https://blog.csdn.net/li_xue_zhao/article/details/79818390>