

# missing semester

看完课程后补上的笔记。首页链接：[计算机教育中缺失的一课 . lecture0.pdf](#)。

视频链接：[\[自制双语字幕\] 计算机教育缺失的一课\(2020\) - 第1讲 - 课程概览与 shell\\_哔哩哔哩\\_bilibili](#)。个人认为胜过youtube版本。

## 目录

<b>1 Lecture1 CourseOverview and Shell</b>	<b>1</b>
1.1 shell基本命令	2
1.2 程序间的流	2
<b>2 Shell Tools and Scripting</b>	<b>3</b>
2.1 Shell 脚本	3
2.2 shell工具	4
<b>3 Editors(Vim)</b>	<b>4</b>
3.1 Vim模式	4
3.2 Vim基础	4
3.3 Vim自定义	5
<b>4 Data Wrangling</b>	<b>5</b>
4.1 正则表达式	6
4.2 其他数据工具	6
4.3 awk	6
<b>5 Command-line Environment</b>	<b>7</b>
5.1 job control	7
5.2 Terminal Multiplexers	7
5.3 dotfiles	8
5.4 Remote Machines	8
<b>6 Version Control(Git)</b>	<b>8</b>
6.1 Git's data model	8
6.2 Staging Area 暂存区	9
6.3 Git command-line interface	9

## 1 Lecture1 CourseOverview and Shell

[课程概览与 shell . lecture 1.pdf](#)

本课程包含 11 个时长在一小时左右的讲座，每一个讲座都会关注一个特定的主题。

教学大部分是面向Linux的。课程会使用bash，最被广泛使用的shell。

当您打开终端时，您会看到一个提示符，它看起来一般是这个样子的：`missing:~$`

这是 shell 最主要的文本接口。它告诉你，你的主机名是 missing 并且您当前的工作目录 (cwd) 或者说您当前所在的位置是 ~ (表示home)。\$符号表示您现在的身份不是 root 用户。

shell 基于空格分割命令并进行解析，然后执行第一个单词代表的程序，并将后续的单词作为程序可以访问的参数。如果您希望传递的参数中包含空格（例如一个名为 My Photos 的文件夹），您要么使用单引号，双引号将其包裹起来，要么使用转义符号 \ 进行处理（`My\ Photos`）。

1.1 shell基本命令

shell 是一个编程环境，所以它具备变量、条件、循环和函数。  
环境变量是在shell启动是设置的东西。其中包含你的主目录路径以及用户名。  
当你执行程序时，shell会在环境变量\$PATH中寻找，直到找到相应的程序。  
可以通过which [arg] 命令来寻找指定程序的路径。

shell 中的路径是一组被分割的目录，在Linux和macOS上使用/分割，而在Windows上是\。  
绝对路径是能完全确定文件位置的路径。。如果某个路径以/开头，那么它是一个绝对路径。  
相对路径是指相对于当前工作目录的路径。  
当前工作目录可以通过pwd(present working directory)获取。  
cd - 会前往上一个目录。

ls -l 可以更加详细地打印出目录下文件或文件夹的信息。  
首先，行中第一个字符d表示该文件是一个目录。然后接下来的九个字符，每三个字符构成一组。它们分别代表了文件所有者，用户组以及其他所有人具有的权限。其中- 表示该用户不具备相应的权限。  
注意，/bin目录下的程序在最后一组，即表示所有人的用户组中，均包含 x 权限，也就是说任何人都可以执行这些程序。  
如果你对于一个文件有写权限，对于目录却没有写权限，那么你能只能清空此文件，不能删除此文件。  
简单来说：

表格 1. rwx

r	是否被允许看到这个目录中的文件
w	是否被允许在该目录中重命名，创建或修改文件
x	是否被允许进入该目录或是执行文件

mv命令可以用于重命名或是移动文件。  
eg: mv aaa bbb将文件 aaa 改名为 bbb。 mv aaa /bbb 将文件aaa移入bbb目录。

cp [origin] [target] 命令可以用于复制文件。 其需要两个参数， 一个是要复制的文件路径，一个是目标路径。

1.2 程序间的流

在 shell 中，程序有两个主要的“流”：它们的输入流和输出流。当程序尝试读取信息时，它们会从输入流中进行读取，当程序打印信息时，它们会将信息输出到输出流中。通常输入是键盘，输出则是屏幕终端，但是我们可以重定向这些流。

最简单的方法是使用<以及>符号。  
例如使用cat < aaa.txt > bbb.txt。此命令会将aaa文件作为cat的输入，同时将cat的输出覆写到bbb文件中。  
>>的作用则是追加内容，而非覆写。  
| 管道(pipes)则可以将一个程序的输出与另一个程序的输入连接起来。

sudo命令可以以superuser的身份执行操作。

tee命令从标准输入中复制到一个文件，并输出到标准输出。  
eg: ping google.com | tee output.txt  
输出内容不仅会被写入文件，也会被显示在终端中。

touch命令可以修改文件的参数，或是创建一个不存在的文件。

## 2 Shell Tools and Scripting

Shell Tools and Scripting · lecture2.pdf

### 2.1 Shell 脚本

一般来说，shell脚本中的空格是用来作为参数分隔的。  
在shell中，\$1表示的是脚本的第一个参数。具体如下：

表格 2. bash特殊符

\$0	脚本名
\$1-\$9	脚本第一到第九个参数
\$@	所有参数
\$#	参数的数量
\$?	上一条指令的返回值
\$\$	现在脚本进程的pid
!!	上一条指令(包括参数)，用来sudo !!执行上一条没执行成功的指令时很有用
\$_	上一条指令的最后一个参数

命令会通过STDOUT返回值，错误则通过STDERR。  
0表示执行正确，其他值则是错误。

short-circuiting<sup>1</sup>逻辑短路指使用计算机中的与或运算来跳过语句。  
eg: `true || echo "will not be printed" and false && echo "will not be printed"`

在shell中，我们可以将一个命令的输出作为一个变量处理。例如 `for file in $(ls)`。  
我们也可以通过 `<(cmd)` 的形式将命令的输出放入一个临时文件中。因为有些命令是从文件中获取输入而非STDIN标准输入，所以有时这很有效。eg: `diff <(ls aaa) <(ls bbb)`。

当在bash中进行逻辑判断时，我们推荐使用 `[[ ]]` 而非 `[ ]`<sup>2</sup>。尽管其不兼容与sh，但这更安全。。

在bash中我们可以通过通配符来简化操作和参数。  
通配符\*与?。\*匹配多位，?只匹配一位。  
eg: `foo1,foo2,foo10. foo?-->foo1,foo2. foo*-->foo1,foo2,foo10`  
花括号则可以用来拓展。  
eg: `mv *.py,.sh` will move all \*.py and \*.sh. `touch {foo,bar}/{a..h}` will create files `foo/a,foo/b...foo/h,bar/a...bar/h`。

你也可以通过其他语言写脚本。

我们可以通过文件顶部的shebang<sup>3</sup>来指定执行脚本的解释器。  
我们可以在shebang中使用env<sup>4</sup>命令来解析系统中相关命令的位置，从而提高可移植性。  
eg: `#!/usr/bin/env python`

### 2.2 shell工具

1. Short-circuit evaluation - Wikipedia.  
2. BashFAQ/031 - Greg's Wiki (woledge.org)  
3. Shebang (Unix) - Wikipedia  
4. env(1) - Linux manual page (man7.org)

man手册可以帮助我们了解命令的使用方法，但有时候man手册提供的内容过多了。这时我们可以通过tldr<sup>5</sup>来了解相关命令的例子从而更好的使用它们。

find命令可以用来寻找文件，同时也可以对这些文件进行操作。

eg: `find . -name '*.tmp' -exec rm {} \;` delete all files with .tmp extension

我们也可以应用更现代且语法更便捷的fd<sup>6</sup>来搜索文件。

locate命令。。感觉没啥用。

搜索文件中的内容可以使用grep命令。你也可以使用更现代的ripgrep。

eg: `rg -u --files-without-match "^#!"`

it will find all files (including hidden files by using -u) without a shebang (by using --files.. to print those who don't match the pattern we give to command)

可以通过history命令来快速的回归命令历史。也可以通过<c-r>来反向搜索命令历史。

fzf<sup>7</sup>是更高效的反向搜索工具。

快速的显现目录结构可以通过ls -R 或是 tree来实现。

## 3 Editors(Vim)

[Editors \(Vim\) . lecture3.pdf](#)

### 3.1 Vim模式

Vim是基于模式的。具体转换如下：

```
normal i ↔esc insert
normal r ↔esc replace
normal v ↔esc visual
normal shift+v ↔esc visual-line
normal <c-v> ↔esc viusal-block
normal : ↔esc command
```

### 3.2 Vim基础

表格 3. vim :

:q	退出当前窗口
:w	保存
:wq	保存并退出
:help {topic}	open help
:e {filename}	切换文件
:ls	显示打开的buffers

Vim会维持一组打开的文件，即 *buffer*。

Vim具有很多的 *tab*, 每一个都对应一个或多个 *window*。

每一个 *window* 显示出一个 *buffer*。

5. tldr pages

6. sharkdp/fd: A simple, fast and user-friendly alternative to 'find' (github.com)

7. Configuring shell key bindings . junegunn/fzf Wiki (github.com)

一个 *buffer* 可以被多个 *window* 显示，即使这些 *window* 在同一个 *tab* 中。

表格 4. Vim movement

w	下一个词 next word
b	单词开头(向前) begining of word
e	单词尾部(向后) end of word
0	行开头
^	此行第一个非空字符
\$	行尾
H	窗口顶部
M	窗口中部
L	窗口底部
<C-u>	向上滚动(up)
<C-d>	向下滚动(down)
gg	文件开头
G	文件尾部
{number}G	跳到相应行
%	在{, [, (上使用会跳转到对应的括号
f{character}	向后跳转到此字符
t{character}	向后跳转到此字符前
F{character}	向前跳转到此字符
T{character}	向前跳转到此字符后
/(word)	使用/进行单词搜索，按下回车后会跳转到相应处

表格 5. Vim modify

~	翻转大小写
ci[	修改[ ]中的内容
da(	删除包括( ) 在内的所有内容

### 3.3 Vim自定义

使用 ~/.vimrc 文件自定义你的 Vim。

你可以在自己的 shell 中启用 vim 模式。

bash 使用 `set -o vi`。zsh 使用 `bindkey -v`。fish 使用 `fish_vi_key_bindings`。

你也可以不论 shell 类型直接使用 `export EDITOR=vim`。但这也会改变其他程序的 editor，比如 git。Readline 也可以使用 `set editing-mode vi` 来进入 Vim 模式。例如 python repl 便会收到影响。

使用 [Vimium - Chrome Web Store \(google.com\)](#) 在 chrome 中启用 Vim 模式。

宏就不介绍了，一时半会说不清楚。

使用最少的操作完成文件处理：[VimGolf - real Vim ninjas count every keystroke!](#)。

## 4 Data Wrangling

[Data Wrangling . lecture4\\_220304\\_002236.pdf](#)

将一种格式的数据变为另一种格式的都可以称为数据整理。

`less` 命令可以用来分页查看文本。

使用 `sed` 这个流编辑器来处理文本。eg: `cat ssh.log | sed 's/.*Disconnected from //'`  
`s` 替换命令的格式是 `s/REGEX/SUBSTITUTION/`

sed除了搜索和替换之外， 其他的功能并不算好。

4.1 正则表达式

正则表达式在默认情况下每行只会匹配一次， 替换一次。默认情况下不会跨行匹配。  
想要全部匹配， 请使用s/REGEX/SUBSTITUTION/g。  
sed中的特殊字符需要使用\来转义。你也可以使用sed -E来使用更现代d1正则表达式。

表格 6.

字符	匹配模式
.	任意字符
*	前一个字符的zero or more
+	前一个字符的one or more
[abc]	a,b,c中的任意一个
(regex1 regex2)	rx1,rx2中的任意一个
^	行开头
\$	行尾

通常来说， \*和+都是greddy贪婪的， 他们会尽可能多的匹配文本。若要改为非贪婪的， 则需要两者后面加上？。  
你可以使用[regex debugger](#)来调试表达式。  
你也可以使用捕获组 capture group来取得正则表达式中匹配上的内容， 只需要用()括起来即可。  
eg: sed -E 's/.\*Disconnected from (invalid |authenticating )?user (.\*?) [^ ]+port [0-9]+( \[preauth\])?\$/\2/'  
其中有三个括号， 我们想要的user后的用户名便是\2。我们将这一整句内容替换为了用户名。

4.2 其他数据工具

sort工具可以对其的输入进行排序。  
sort -n将会以第一列的数字大小排序而非字典序。 -k则以选择输入中以空格为分隔符的列来执行排序。sort -nk1,1即从第一列开始， 在第一列结束。sort -r可以倒序输出。

uniq工具会查看一个排序后的行列表， 然后其会去除那些重复的行。多个相同的行只会打印一次。  
uniq -c将计算任何重复行的次数并消除他们， 并将次数加到行前缀上。

可以使用tail -n5， head -n10类似的工具获取输出的头部和尾部。

paste命令可以将多个行合并在一起， 使其合为一行。-s会按输入将输出的一行分隔， -d则会指定分隔符(eg: paste -sd,)。

可以使用wc -l计算行的数量。使用bc来计算表达式。  
比如说将多行数字输出使用|paste -sd+ | bc -l 将多行数字输出使用+连接后使用bc计算。  
使用R进行数据统计， 使用gnuplot画图。

xargs可以将一系列输入转化为参数,很有用。比如你有很多旧版本的工具链需要删除时。  
eg: xxx toolchain |grep 2019| xargs xxx toolchain uninstall

4.3 awk

awk也是处理文件流的， 但其更专注与处理列。其默认将输入解析为以空格为分隔符的列。  
在awk参数中， \$0指全部内容， \$1-n则指1-n列。

eg: `awk ' $1 ==1 && $2 ~ /^c.*e$/ {print $2}'`  
 输出所有第一列为1且第二列匹配模式的行的第二列。

awk编程,输出有多少行匹配条件的:  
`BEGIN {rows = 0}`  
`$1 ==1 && $2 ~ /^c.*e$/ {rows +=1 }`  
`END {print rows}`

begin匹配输入开头, end匹配输入尾。

## 5 Command-line Environment

[command-line environment . lecture5.pdf](#)

### 5.1 job control

<C-c> 停止当前进程是因为终端向程序发送了一个SIGINT信号。该信号表示程序中断。  
 我们也可以使用<C-\>来发送SIGQUIT信号来终止程序。中断和终止是不同的。  
 SIGKILL无法被程序捕获, 如果发送SIGKILL, 它无论如何都将终止进程的执行。

通过<C-z>发送SIGSTOP信号可以将程序暂停。  
 使已经暂停的在后台进程再次运行可以使用bg %n。n指其在Jobs中的序号。  
 在命令后加上&会使命令在后台运行。

我们可以使用kill命令发送任何信号。eg: `kill -STOP %1`停止选定进程。  
 nohup命令将所执行的命令封装起来, 忽略任何挂起信号, 并使其继续运行。

### 5.2 Terminal Multiplexers

推荐将<C-b>重映射为<C-a>, 反正你自己已经在dotfiles中设置好了。

tmux的几个核心概念:  
 -sessions: 有多个windows

tmux -ls会显示当前所有tmux sessions  
 tmux new -s NAME 创建带名字的session  
 可以使用<C-b> d 脱离此session  
 tmux -a 重新回到上一个session, 可以使用-t指定回归的session

-windows: 类似于编辑器和浏览器中的标签。

<C-b> c可以创建一个新窗口。关闭只需用<C-d>即可  
 <C-b> p前往上一个窗口, <C-b> n前往下一个窗口。  
 <C-b> ,可以更改窗口名。<C-b> w显现出所有现在的窗口。  
 <C-b> N可以跳转至N号窗口。

-panes: 分裂窗口

<C-b> "垂直分裂窗口。<C-b> %水平分裂窗口。  
 <C-b> 箭头用来在panes间按方向跳动。  
 <C-b> 扩张当前窗口。

`<C-b> space` 改变当前panes的排列。

### 5.3 dotfiles

alisa别名可以省去很多时候输入命令参数的麻烦。

eg: `alisa ll="ls -lh"` 由于alias是shell命令，所以=旁边不能有空格。

`mv=mv -i mkdir=mkdir -p`都是可以尝试的。

有许多很关键的配置文件，例如.vimrc, .bashrc可以放置在dotfiles中。

我们使用dotfiles下的安装脚本建立软链接，从而快捷的安装所有的配置文件。dotfiles可以通过git获取，而链接可以通过ln -s创建。可以参见你自己的dotfiles中的install.sh明晰安装过程。

你可以上网学习其他人的dotfiles中的配置并化为己用。

### 5.4 Remote Machines

ssh为其中关键。

一般来说使用ssh username@ip来进行连接。

每次都要输入密码是有点麻烦的，我们可以使用公钥私钥体系。

可以先使用`ssh-keygen -o -a 100 -t ed25519 -f ~/.ssh/id_ed25519`创建keys。

当然，你在使用github时可能配置过了。

运行`cat ~/.ssh/id_ed25519.pub | ssh username@ip 'cat >> ~/.ssh/authorized_keys`

可以使用更简单的`ssh-copy-id -i ~/.ssh/id_ed25519.pub username@ip`。其会复制密钥。

在远端复制文件不能使用cp，但可以使用scp命令。

使用rsync复制则是更为先进。

你可以在./ssh/config文件里配置内容。

```
Host vm
  User shulva
  HostName 111.111.111.111
  Port 2222
  IdentityFile ~/.ssh/id_ed25519
```

之后使用ssh vm即可登录。

在config里配置，便利性和可移植性都有所提高。你也可以将自己的ssh文件放在dotfiles中。

但这会对安全性有所影响。

## 6 Version Control(Git)

[Version Control \(Git\) . lecture6.pdf](#)

### 6.1 Git's data model

在git数据模型中，一个文件被称之为blob，一堆字节的合集。而一个文件夹则被称之为tree。tree中可以包含一系列tree和blob。

一个git的快照snapshot本质上是对最顶层的tree(directory)的追踪。

git使用DAG来处理快照间的关系。一般来说git将这些快照称为commit。



git中的commit是不可改变的，但这并不意味着错误是不可更改的。但是这种更改会产生新的commit。你可以添加新的东西，但是无法改动图的结构。

git的数据模型伪代码示例如下：

```
type blob = array<byte>
type tree = map<string,tree|blob>
type commit = struct{
    parent:array<commit>
    author:string
    message:string
    snapshot:tree
}
```

git数据模型实际磁盘存储的数据结构如下,使用hash函数记录：

```
type object = blob|tree|commit
objects = map<string,object>
```

```
def store(objects):
    id = sha1(object)
    object[id]=object
```

```
def load(objects):
    return objects[id]
```

blob,tree,commit都是Objects。当他们指向其他objects时，他们并不是真的包含Objects。他们只是拥有他们的引用。

所有object都储存于Object表中，git中不同对象之间的引用都是通过它们的id，也即是hash值。

所有的快照都可以被它们的SHA-1 hash值唯一标识。但是hash值是一组长为40的字符串，不好记忆。所以git通过维护references来解决这一问题。

git的引用模型伪代码示例如下：

```
references = map<string1_name,string2_hash>
通过自己可记忆的名称来映射hash值。
```

## 6.2 Staging Area 暂存区

暂存区会告知git在下次创建快照/commit时应该包含哪些更改。

git将追踪文件的权力交给了用户，从而让用户可以自己定义快照，而非直接快照整个文件。

使用git add <filename>可以直接将文件添加到暂存区。

## 6.3 Git command-line interface

推荐Git - Book ([git-scm.com](http://git-scm.com))了解具体细节。

git commit的信息编写：

[How to Write a Git Commit Message \(cbea.ms\)](http://cbea.ms)

[tbagery - A Note About Git Commit Messages](http://tbagery.com)

HEAD指向最后一个快照。

git log --all --graph --decorate 将提交历史用DAG的方式显现出来（其实也没显示多少）

git log --all --graph --decorate --oneline 更精简的图结构

git checkout <reference> 更新HEAD到你指向的commit或是分支，改变你当前工作目录

git diff可以显示当前目录与上次快照之间的不同

git diff <filename>当前文件与上次快照间的不同

```
git diff <reference> <filename> 当前工作目录与指定快照间文件的不同
git diff <reference> <reference> <filename> 两个指定快照间文件的不同
```

```
git branch -vv 会列出在本地仓库的所有分支, -vv 是详细信息。
git branch <name> 创建新分支。
git checkout -b <name> 创建并切换到新分支
```

```
git merge <name> 将此分支合并到当前分支
git mergetool 启用专门工具处理合并冲突
```

一般来说冲突文件中会包含如下内容:

```
<<<<< HEAD
... 当前分支的
=====
... 合并分支的
>>>>> branch name
```

```
git remote 可以列出当前仓库所知道的所有远程仓库。
git remote add <name> <url> 添加一个远程仓库。
git push <remote> <local branch>:<remote branch>
通过本地分支创建远程仓库上的分支。
git branch --set-upstream-to=<remote>/<remote branch> 将本地分支与远程分支相对应。
这会简化git pull/push, 不用输入大量参数。
git fetch 获取远端更改, 从远端获取对象和引用, 不会更改任何本地历史记录。
git pull= git fetch+git merge
```

```
git config 自定义, 自己修改~/.gitconfig亦可。
git clone --depth=1 不会clone历史提交记录
git add -p 交互式的暂存
git blame 显示每一行是谁修改提交的
git show 显示提交信息
git stash 暂时移除更改 git stash pop是相对应的逆过程
git bisect (binary search history?)
.gitignore 使用gitignore文件指定无需上传的文件类型 eg:*.DS_Store
```

```
git commit --amend 修改一个commit的内容和信息
git reset HEAD <file>:反暂存文件
git checkout -- <file>:舍弃修改
```