工欲善其事 必先利其器

软件开发中的常用工具

孟宁

关注孟宁

软件开发中的常用工具

- Visual Studio Code & Vim
- find & grep
- Git
- Regular Expression

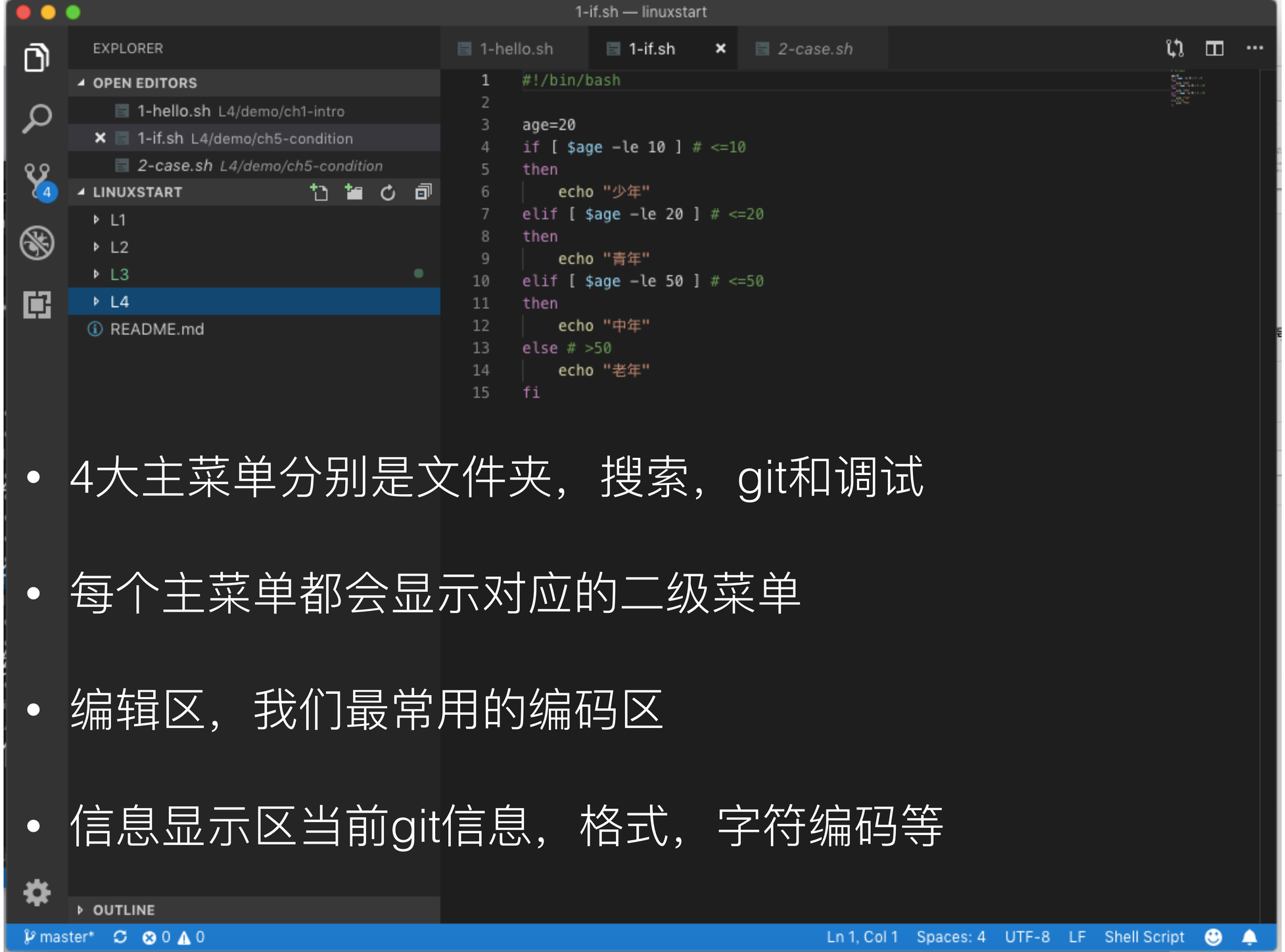
Visual Studio Code

- Visual Studio Code（以下简称vscode）是一个轻量且强大的代码编辑器，支持Windows，OS X和Linux。内置JavaScript、TypeScript和Node.js支持，而且拥有丰富的插件生态系统，可通过安装插件来支持C++、C#、Python、PHP等其他语言。

- https://code.visualstudio.com/#alt-downloads

- sudo apt install ./<file>.deb

Visual Studio Code alt-downloads



- 4大主菜单分别是文件夹，搜索，git和调试
- 每个主菜单都会显示对应的二级菜单
- 编辑区，我们最常用的编码区
- 信息显示区当前git信息，格式，字符编码等

VS Code 为什么能这么牛?

- VS Code近年来获得了爆炸式增长，成为广大开发者工具库中的必备神器。VS Code 为什么能这么牛?
  - 简洁而聚焦的产品定位，贯穿始终
  - 进程隔离的插件模型
  - UI 渲染与业务逻辑隔离，一致的用户体验
  - 代码理解和调试——LSP和DAP两大协议
  - 集大成的 Remote Development

VS Code简洁而聚焦的产品定位

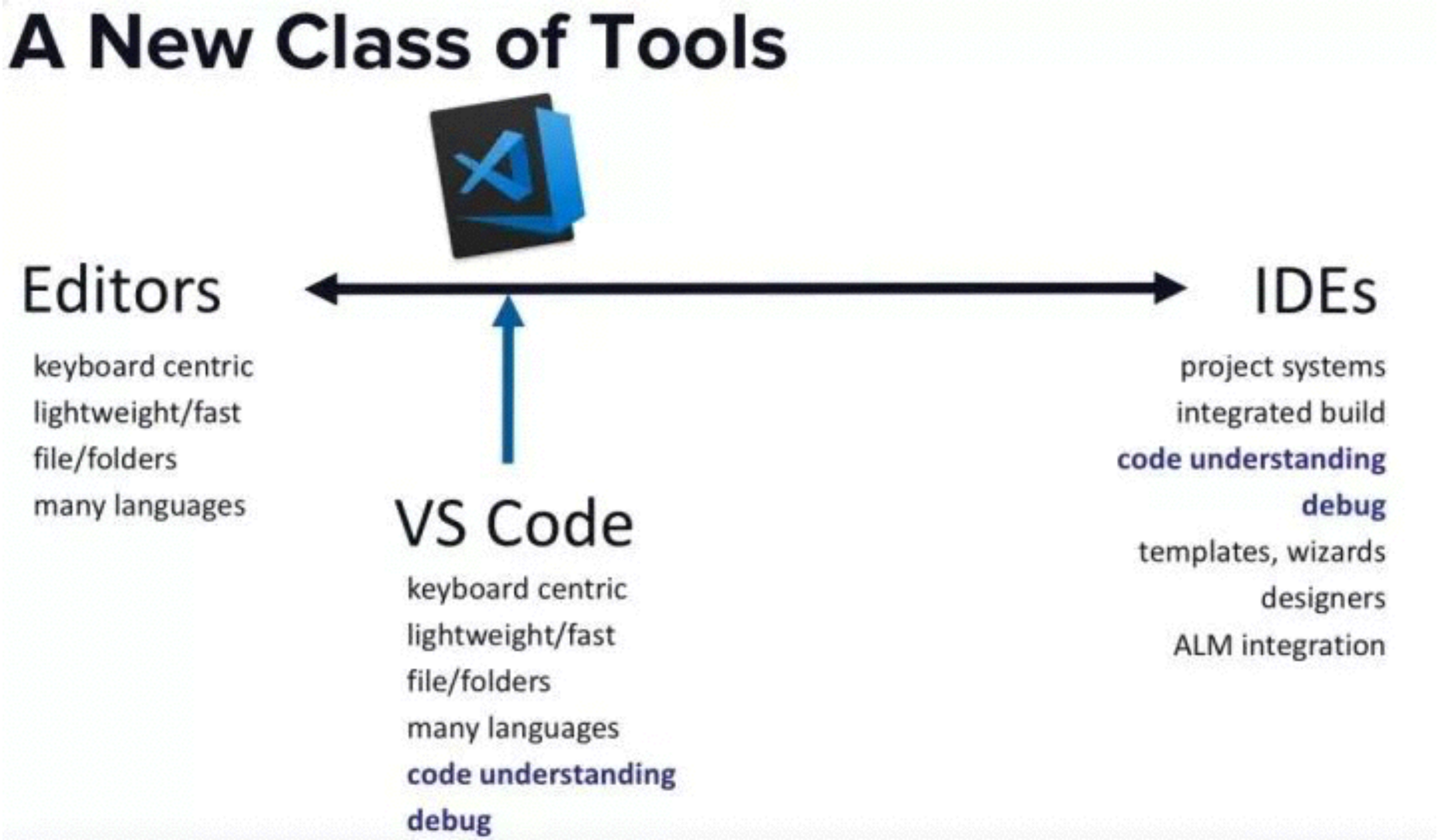
- 你知道 VS Code 的开发团队人数只有二十出头吗？难以相信吧，大家都觉得 VS Code 无所不能，如此强大的工具那么几个人怎么做得出来。实际上功能丰富是个美好的错觉，因为大部分针对特定编程语言和技术的功能都是第三方插件提供的，VS Code 的核心始终非常精简，这很考验产品团队的拿捏能力：**做多了，臃肿，人手也不够；做少了，太弱，没人用。**

- **“简洁”说到底是产品的“形态”，更关键的其实是前置问题——产品的定位，它到底解决什么问题。**

  - **我们为什么需要一个新的工具?**

  - **它到底是代码编辑器(Editor)还是集成开发环境(IDE)?**

项目负责人Erich Gamma 的说法



专注于开发者“最常用”的功能

- 编辑器+代码理解+调试。这是一个非常节制而平衡的选择，专注于开发者“最常用”的功能，同时在产品的形式上力求简洁高效。从结果来看，这个定位是相当成功的。

- 相对较小的功能集，使得开发者们能在代码质量上精益求精，最终用户们也得到了一个性能优异的工具，这是 VS Code 从一众编辑器中脱颖而出的重要原因。

- 正因为产品定位以及团队职责上的高度节制，团队成员才能把时间花在这类问题上，写出经得起考验的代码。较小的团队也使得团队成员做到了行为层面的整齐划一。

进程隔离的插件模型

- 码农千千万，你用 Node 我用 Go，你搞前端我弄后台，VS Code 如何满足这些五花八门的需求呢？机智的你已经抢答了——海量插件。
- 通过插件来扩展功能的做法已经是司空见惯了，但如何保证插件和原生功能一样优秀呢？历史告诉我们：不能保证：）。
- Eclipse插件模型可以说是做得非常彻底了，功能层面也是无所不能，但存在几个烦人的问题：不稳定、难用、慢，所以不少用户转投 IntelliJ 的怀抱。可谓成也插件，败也插件。

进程隔离的插件模型

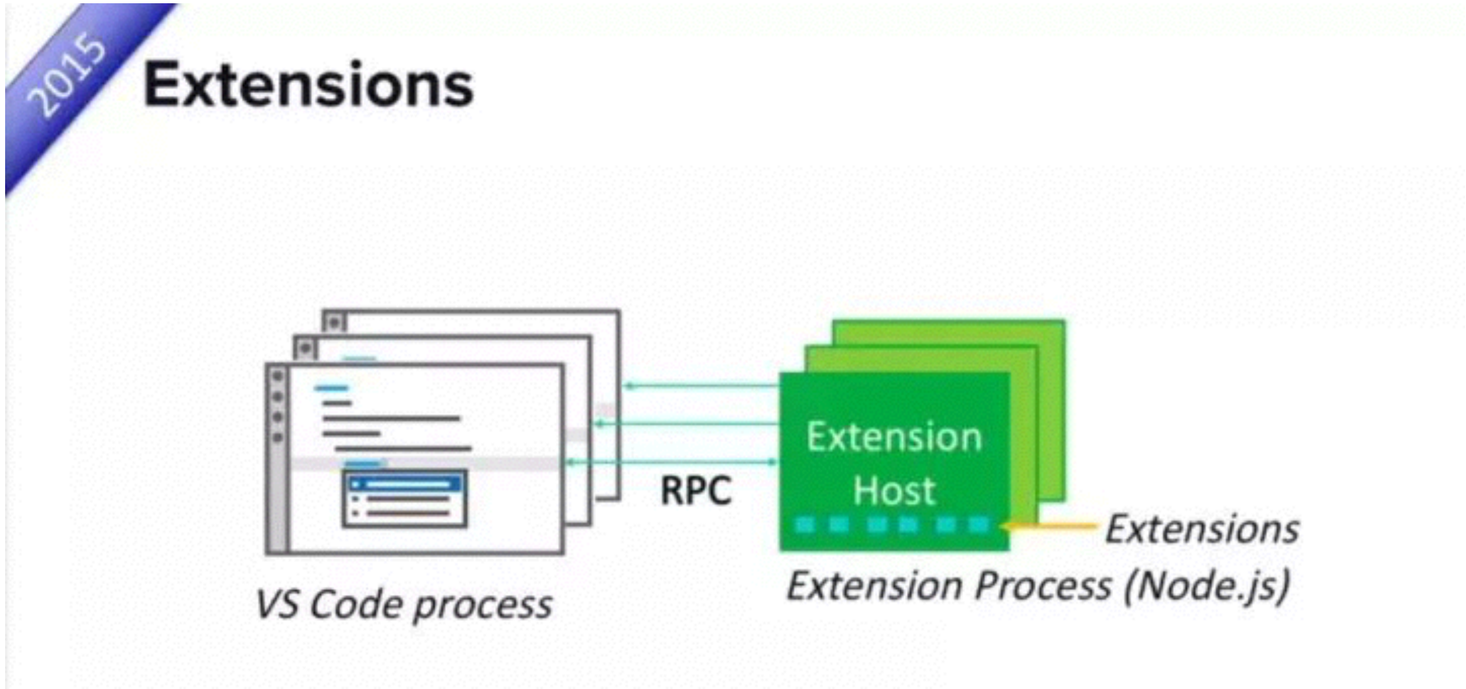
- 不同团队写出来的代码，无论是思路还是质量，都不一致。最终，用户得到了一个又乱又卡的产品。所以要让插件在稳定性、速度和体验的层面都做到和原生功能统一，只能是一个美好的愿望。

- Visual Studio 自己搞定所有功能，并且做到优秀，让别人无事可做，这也成就了其“宇宙第一IDE”的美名；IntelliJ 与之相仿，开箱即用，插件可有可无。这么看起来，自己搞定所有的事情是个好办法，但大家是否知道，Visual Studio 背后有上千人的工程团队，

- Eclipse 核心部分的开发者就是早期的 VS Code 团队。嗯，所以他们没有两次踏入同一条河流。与 Eclipse 不同， VS Code 选择了把插件关进笼子里。

进程隔离的插件模型

- 稳定性对于 VS Code 来说尤为重要。都知道 VS Code 基于 Electron，实质上是个 Node.js 环境，单线程，任何代码崩了都是灾难性后果。所以 VS Code 干脆不信任任何人，把插件们放到单独的进程里，任你折腾，主程序妥妥的。



UI 渲染与业务逻辑隔离，一致的用户体验

- “不稳定”之后的问题是“难用”，具体来说就是混乱的界面和流程，究其原因就是插件之间的界面语言的“不一致”，它导致学习曲线异常陡峭，并且在面临问题时没有统一的解决路径。VS Code 的做法是根本不给插件们“发明”新界面的机会。

- VS Code 统管所有用户交互入口，制定交互的标准，所有用户的操作被转化为各种请求发送给插件，插件能做的就是响应这些请求，专注于业务逻辑。但从始至终，插件都不能“决定”或者“影响”界面元素如何被渲染（颜色、字体等，一概不行），至于弹对话框什么的，就更是天方夜谭了。

代码理解和调试——LSP和DAP两大协议

- 代码理解和调试，绝大部分都由第三方插件来实现，中间的桥梁就是两大协议——Language Server Protocol(LSP)和 Debug Adapter Protocol(DAP)。

- 全栈开发早已成为这个时代的主流，软件从业者们也越来越不被某个特定的语言或者技术所局限，这也对我们手里的金刚钻提出了新的挑战。举个栗子，我用 TypeScript 和 Node.js 做前端，同时用 Java 写后台，偶尔也用 Python 做一些数据分析，那么我很有可能需要若干工具的组合，这样做的问题就在于需要在工具间频繁切换，无论从系统资源消耗和用户体验的角度来看，都是低效的。

- 那么有没有一种工具能在同一个工作区里把三个语言都搞定呢？没错，就是 VS Code——支持多语言的开发环境，而多语言支持的基础就是Language Server Protocol(LSP)。

LSP协议

- Language Server Protocol(LSP)协议在短短几年内取得了空前的成功，到目前为止，已经有来自微软等大厂以及社区的一百个实现，基本覆盖了所有主流编程语言。同时，它也被其他开发工具所采纳，比如 Atom、Vim、Sublime、Emacs、Visual Studio 和 Eclipse，从另一个角度证明了它的优秀。

- 更难能可贵的是，该协议还做到了轻量和快速，可以说是 VS Code 的杀手级特性了，同时也是微软最重要的 IP 之一。。。哇塞，又强大又轻巧，怎么看都是个骗局啊，那我们就来看看它到底怎么做到的。

- 先划重点：**1、节制的设计 2、合理的抽象 3、周全的细节。**

追求大而全是很常见的问题

- 先来说说设计(Design)，大而全是很常见的问题。如果让我来设计这么一个用来支持所有编程语言的东西，第一反应很可能是搞个涵盖所有语言特性的超集。

- 微软就有过这样的尝试，比如 Roslyn——一个语言中立的编译器，C# 和 VB.NET 的编译器都是基于它做的。大家都知道 C# 在语言特性层面是非常丰富的，Roslyn 能撑起 C# 足以说明它的强大。

- 那么问题来了，为啥它没有在社区得到广泛应用呢？我想根本原因是“强大”所带来的副作用：复杂、主观(Opinionated)。光是语法树就已经很复杂了，其他各种特性以及他们之间的关系更是让人望而却步，这样一个庞然大物，普通开发者是不会轻易去碰的。

节制的设计

- LSP 显然把小巧作为设计目标之一，它选择做最小子集，贯彻了团队一贯节制的作风。它关心的是用户在编辑代码时最经常处理的物理实体（比如文件、目录）和状态（光标位置）。它根本没有试图去理解语言的特性，编译也不是它所关心的问题，所以自然不会涉及语法树一类的复杂概念。

- 小归小，功能可不能少，所以抽象就非常关键了。LSP 最重要的概念是动作和位置，LSP 的大部分请求都是在表达”在指定位置执行规定动作“。抽象成请求(Request)和回复(Response)，同时规定了它们的规格(Schema)，在开发者看来，概念非常少，交互形式也很简单，实现起来非常轻松。

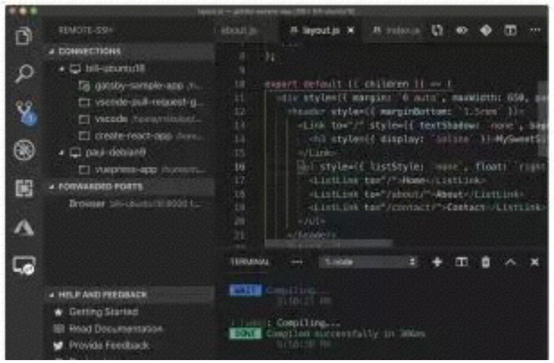
做设计的时候一定要倾向于简单

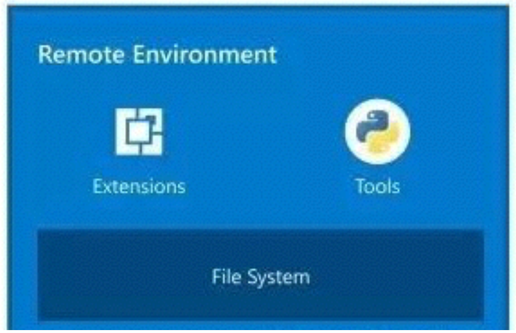
- 首先这是一个基于文本的协议，文本降低了理解和调试的难度。参考 HTTP 和 REST 的成功，很难想象如果这是一个二进制协议会是什么局面，甚至同样是文本协议的 SOAP 也早已作古，足以说明“简单”在打造开发者生态里的重要性。

- 其次这是一个基于 JSON 的协议，JSON 可以说是最易读的结构化数据格式了。大家看看各个代码仓库里的配置文件都是啥格式就知道这是个多么正确的决定了，现在还有人在新项目里用 XML 吗？又一次——“简单”。

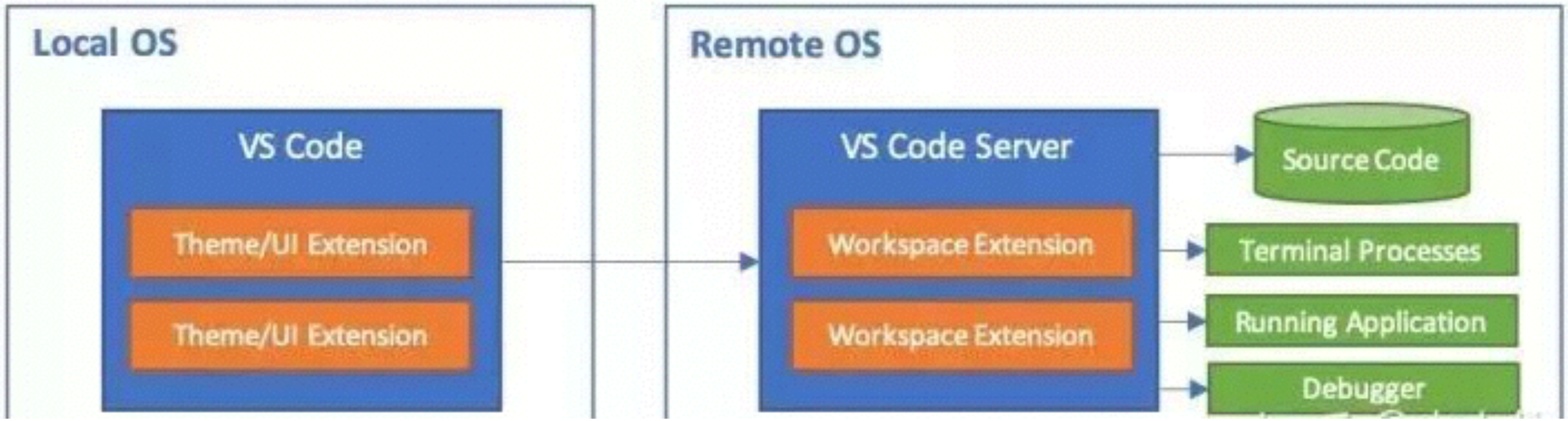
- 再次，这是一个基于 JSONRPC 的协议。由于 JSON 的流行，各大语言都对它有极好的支持，所以开发者根本不需要处理序列化、反序列化一类的问题，这是实现层面的“简单”。

Remote Development(VSCRD)





Remote Development(VSCRD)可以在远程环境（比如虚机、容器）里开一个 VS Code 工作区，然后用本地的 VS Code 连上去工作



VSCRD

- 响应迅速：VSCRD 所有的交互都在本地 UI 内完成，响应迅速；远程桌面由于传输的是截屏画面，数据往返延迟很大，卡顿是常态。
- 沿用本地设置：VSCRD 的 UI 运行在本地，遵从所有本地设置，所以你依然可以使用自己所习惯的快捷键、布局、字体，避免了工作效率层面的开销。
- 数据传输开销小：远程桌面传输的是视频数据，而 VS Code 传输是操作请求和响应，开销与命令行相仿，卡顿的情况进一步改善。
- 第三方插件可用：在远程工作区里，不仅VS Code的原生功能可用，所有第三方插件的功能依然可用；远程桌面的话，你得自己一个个装好。
- 远程文件系统可用：远程文件系统被完整映射到本地，这个两者差不多。
- VSCRD 做了什么神奇的操作能够实现以上效果呢?

前瞻性的架构决策和扎实的工程基础

- 进程级别隔离的插件模型。Extension Host（也就是VS Code Server）与主程序做到了物理级别的分离，那么把 Extension Host 在远程或者本地跑没有本质的区别。
- UI 渲染与插件逻辑隔离，整齐划一的插件行为。所有的插件的 UI 都由 VS Code 统一渲染，所以插件里面只有纯业务逻辑，行为高度统一，跑在哪里都没区别。
- 高效的协议LSP。VS Code 的两大协议 LSP、DAP 都非常精简，天然适合网络延迟高的情况，用在远程开发上再适合不过。
- VS Code 团队在架构上的决策无疑是非常有前瞻性的，与此同时，他们对细节的把握也是无可挑剔。正因为有了如此扎实的工程基础，VSCRD 这样的功能才得以诞生，所以被认为这是集大成的作品。

VSCRD 非常有用的场景

- 开发环境配置起来很繁琐，比如物联网开发，需要自己安装和配置各种工具和插件。在 VSCRD 里，一个远程工作区的模板即可搞定

- 本地机器太弱，某些开发搞不了，比如机器学习，海量数据及计算需求需要非常好的机器。在 VSCRD 里，可以直接操作远程文件系统，使用远程计算资源。

VS Code 为什么能这么牛?

- VS Code 像一颗耀眼的星星，吸引着成千上万开发者为其添砖加瓦。
- 从 VS Code 的成功中，我们看到了好的设计和工程实践能创造奇迹。放眼软件产业，各个层面的模式不断被刷新，让人激动之余，也要求从业者不断提高技能水平。
- 从个人学习的角度来看，了解这些模式诞生的前因后果，理解工程实践中的决策过程是非常有利于提高工程能力的。

VS Code带来的一点启发和思考

- 1980s-1990s，开工厂，苏南乡镇企业、浙江民营企业…
- 2000s-2010s，渠道为王-模式创新，义乌小商品市场、阿里巴巴/淘宝、腾讯、百度…，P2P、消费贷、金融租赁、共享经济…
- 2020s-，产品为王，会有越来越多的优秀产品像VS Code一样在一个成熟的充分竞争的市场异军突起。

Vim - the ubiquitous text editor

孟宁

关注孟宁

Linux vi/vim

- 几乎所有的Unix-Like系统一般都会预装vi文本编辑器，其他的文本编辑器则不一定预装。
- vim具有程序编辑的能力，可以主动的以字体颜色辨别语法的正确性，方便程序设计。
- vim是从vi发展出来的一个文本编辑器。代码补完、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用。
- 简单的来说，vi仅仅是文本编辑器，不过功能已经很齐全了。 vim则是程序开发者的一项很好用的工具。连vim的官方网站 (http://www.vim.org) 也说vim是一个程序开发工具而不仅是文本编辑器。

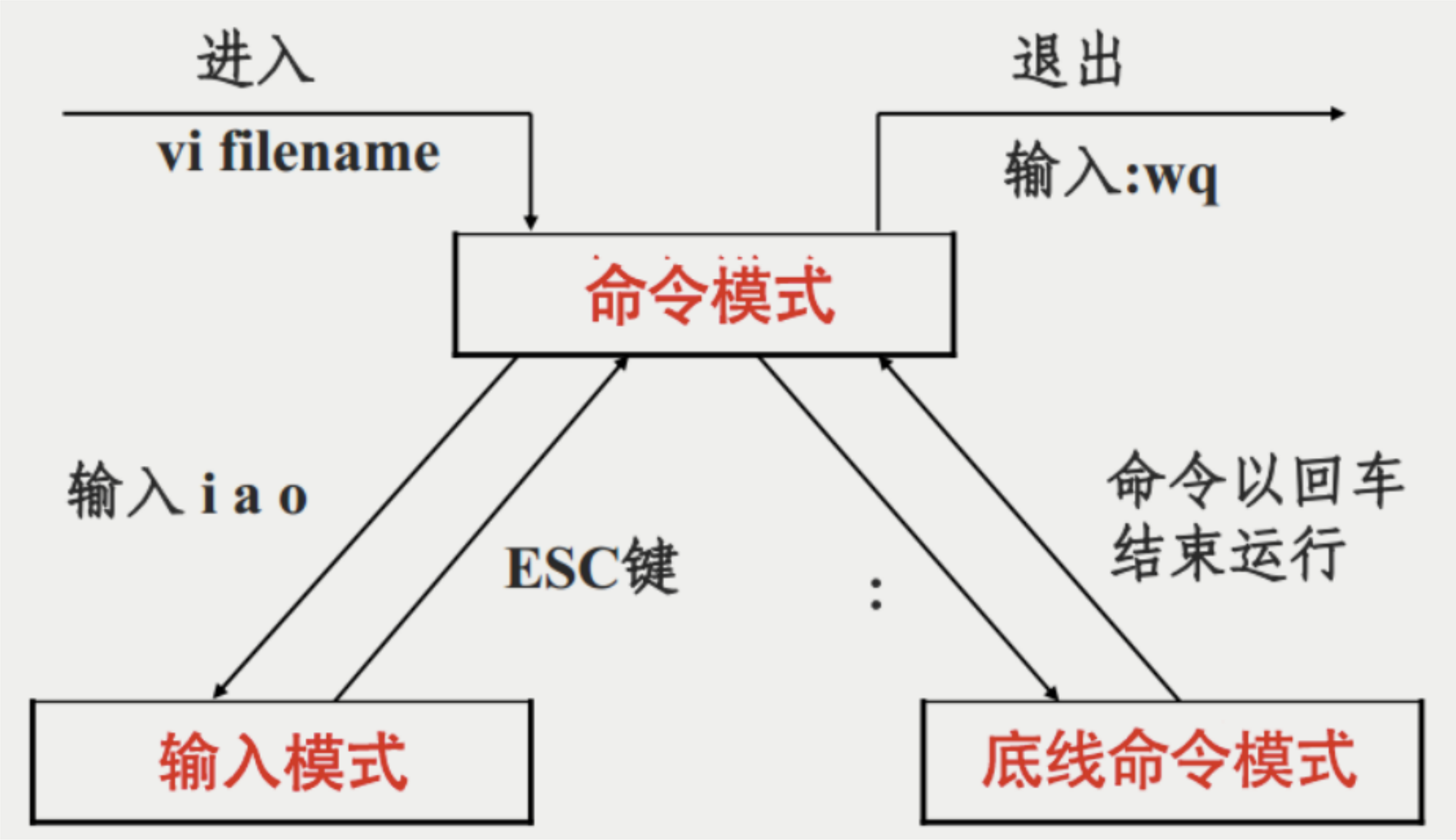
vi/vim的三种模式

- 命令模式（Command mode），用户刚刚启动vi/vim，便进入了命令模式。此状态下敲击键盘动作会被vim识别为命令，而非输入字符。比如我们此时按下i，并不会输入一个字符，i被当作了一个命令。命令模式只有一些最基本的命令，因此仍要依靠底线命令模式输入更多命令

- 输入模式（Insert mode），在命令模式下按下i就进入了输入模式，按ESC退出输入模式，切换到命令模式。

- 底线命令模式（Last line mode），在命令模式下按下:（英文冒号）就进入了底线命令模式。底线命令模式可以输入单个或多个字符的命令，可用的命令非常多。基本的命令有q（退出程序）、w（保存文件）等。按ESC键可随时退出底线命令模式。

vi/vim的三种模式


进入
vi filename
退出
输入:wq
命令模式
输入 i a o
ESC键
:
命令以回车
结束运行
输入模式
底线命令模式

移动光标的方法

- h 或 向左箭头键(←)　光标向左移动一个字符
- j 或 向下箭头键(↓)　　光标向下移动一个字符
- k 或 向上箭头键(↑)　　光标向上移动一个字符
- l 或 向右箭头键(→)　　光标向右移动一个字符
- + 光标移动到非空格符的下一行
- - 光标移动到非空格符的上一行
- 如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。 如果想要进行多次移动的话，例如向下移动 30 行，可以使用 "30j" 或 "30↓" 的组合按键， 亦即加上想要进行的次数(数字)后，按下动作即可！

移动光标的方法

- n<space>　那个 n 表示『数字』，例如 20 。按下数字后再按空格键，光标会向右移动这一行的 n 个字符。例如 20<space> 则光标会向后面移动 20 个字符距离。
- 0 或功能键[Home] 这是数字『 0 』： 移动到这一行的最前面字符处 (常用)
- $ 或功能键[End]　移动到这一行的最后面字符处(常用)
- H　光标移动到这个屏幕的最上方那一行的第一个字符
- M　光标移动到这个屏幕的中央那一行的第一个字符
- L　光标移动到这个屏幕的最下方那一行的第一个字符
- G　移动到这个档案的最后一行(常用)
- nG n为数字。移动到这个档案的第 n 行。例如 20G 则会移动到这个档案的第 20 行(可配合 :set nu)
- gg 移动到这个档案的第一行，相当于 1G 啊！ (常用)
- n<Enter>　n 为数字。光标向下移动 n 行(常用)

翻页

- [Ctrl] + [f]　屏幕『向下』移动一页，相当于 [Page Down]按键 (常用)
- [Ctrl] + [b]　屏幕『向上』移动一页，相当于 [Page Up] 按键 (常用)
- [Ctrl] + [d]　屏幕『向下』移动半页
- [Ctrl] + [u]　屏幕『向上』移动半页

删除、复制与粘贴

- x, X　在一行字当中，x 为向后删除一个字符 (相当于 [del] 按键)， X 为向前删除一个字符(相当于 [backspace] 亦即是退格键) (常用)
- nx　n 为数字，连续向后删除 n 个字符。举例来说，我要连续删除 10 个字符，『10x』。
- dd　删除游标所在的那一整行(常用)
- ndd　n 为数字。删除光标所在的向下 n 行，例如 20dd 则是删除 20 行 (常用)
- d1G　删除光标所在到第一行的所有数据
- dG　删除光标所在到最后一行的所有数据
- d$　删除游标所在处，到该行的最后一个字符
- d0　那个是数字的0 ，删除游标所在处，到该行的最前面一个字符

删除、复制与粘贴

- yy　复制游标所在的那一行(常用)
- nyy n 为数字。复制光标所在的向下 n 行，例如 20yy 则是复制 20 行(常用)
- y1G　　复制游标所在行到第一行的所有数据
- yG 复制游标所在行到最后一行的所有数据
- y0　复制光标所在的那个字符到该行行首的所有数据
- y$　复制光标所在的那个字符到该行行尾的所有数据

删除、复制与粘贴

- p, P　p为将已复制的数据在光标下一行贴上，P则为贴在游标上一行！ 举例来说，我目前光标在第 20 行，且已经复制了 10 行数据。则按下 p 后， 那 10 行数据会贴在原本的 20 行之后，亦即由 21 行开始贴。但如果是按下 P 呢？ 那么原本的第 20 行会被推到变成 30 行。 (常用)
- J 将光标所在行与下一行的数据结合成同一行
- c 重复删除多个数据，例如向下删除 10 行，[10cj]
- u 复原前一个动作。(常用)
- [Ctrl]+r　重做上一个动作。(常用)
- 这个 u 与 [Ctrl]+r 是很常用的指令！一个是复原，另一个则是重做一次

搜索替换

- /word　向光标之下寻找一个名称为 word 的字符串。例如要在档案内搜寻 vbird 这个字符串，就输入 /vbird 即可！ (常用)
- ?word　向光标之上寻找一个字符串名称为 word 的字符串。
- n这个 n 是英文按键。代表重复前一个搜寻的动作。举例来说， 如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字符串，则按下 n 后，会向下继续搜寻下一个名称为 vbird 的字符串。如果是执行 ?vbird 的话，那么按下 n 则会向上继续搜寻名称为 vbird 的字符串！
- N　这个 N 是英文按键。与 n 刚好相反，为『反向』进行前一个搜寻动作。 例如 /vbird 后，按下 N 则表示『向上』搜寻 vbird 。
- 使用 /word 配合 n 及 N 是非常有帮助的！可以让你重复的找到一些你搜寻的关键词！

搜索替换

- :n1,n2s/word1/word2/g n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串取代为 word2！举例来说，在 100 到 200 行之间搜寻 vbird 并取代为 VBIRD 则：

- 『:100,200s/vbird/VBIRD/g』。(常用)

- :1,$s/word1/word2/g 或 :%s/word1/word2/g从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2！(常用)

- :1,$s/word1/word2/gc 或 :%s/word1/word2/gc 从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2！且在取代前显示提示字符给用户确认 (confirm) 是否需要取代！(常用)

切换到编辑模式

- i, I 进入输入模式(Insert mode)：
- i 为『从目前光标所在处输入』， I 为『在目前所在行的第一个非空格符处开始输入』。(常用)
- a, A　　进入输入模式(Insert mode)：
- a 为『从目前光标所在的下一个字符处开始输入』， A 为『从光标所在行的最后一个字符处开始输入』。(常用)
- o, O　　进入输入模式(Insert mode)：
- 这是英文字母 o 的大小写。o 为『在目前光标所在的下一行处输入新的一行』； O 为在目前光标所在处的上一行输入新的一行！(常用)
- r, R　　进入取代模式(Replace mode)：
- r 只会取代光标所在的那一个字符一次；R会一直取代光标所在的文字，直到按下 ESC 为止；(常用)
- [Esc]　退出编辑模式，回到一般模式中(常用)
- 编辑模式在vi画面的左下角处会出现『--INSERT--』或『--REPLACE--』的字样

命令模式

- :w　将编辑的数据写入硬盘档案中(常用)
- :w!　若文件属性为『只读』时，强制写入该档案。不过，到底能不能写入， 还是跟你对该档案的档案权限有关啊!
- :q　离开 vi (常用)
- :q!　若曾修改过档案，又不想储存，使用！为强制离开不储存档案。
- 注意一下啊，那个惊叹号 (!) 在 vi 当中，常常具有『强制』的意思～
- :wq　储存后离开，若为 :wq! 则为强制储存后离开 (常用)
- ZZ　这是大写的 Z 喔！若档案没有更动，则不储存离开，若档案已经被更动过，则储存后离开!

命令模式

- :w [filename]　将编辑的数据储存成另一个档案（类似另存新档）
- :r [filename]　在编辑的数据中，读入另一个档案的数据。亦即将『filename』这个档案内容加到游标所在行后面
- :n1,n2 w [filename] 将 n1 到 n2 的内容储存成 filename 这个档案。
- :! command　暂时离开 vi 到指令行模式下执行 command 的显示结果！例如『:! ls /home』即可在 vi 当中察看 /home 底下以 ls 输出的档案信息！

vim环境的变更

- :set nu　显示行号，设定之后，会在每一行的前缀显示该行的行号
- :set nonu　与 set nu 相反，为取消行号！

代码中批量添加注释

- 批量注释：Ctrl + v 进入块选择模式，然后移动光标选中你要注释的行，再按大写的 I 进入行首插入模式输入注释符号如 // 或 #，输入完毕之后，按两下 ESC，Vim 会自动将你选中的所有行首都加上注释，保存退出完成注释。

- 取消注释：Ctrl + v 进入块选择模式，选中你要删除的行首的注释符号，注意 // 要选中两个，选好之后按 d 即可删除注释，ESC 保存退出。

- 批量注释：使用下面命令在指定的行首添加注释。使用名命令格式： :起始行号,结束行号s/^/注释符/g（注意冒号），如:10,20s#^#//#g，:10,20s/^/#/g

- 取消注释：使用名命令格式： :起始行号,结束行号s/^注释符//g（注意冒号），如:10,20s#^//##g，:10,20s/#//g

version 1.1
April 1st, 06
翻译:2006-5-21

vi / vim 键盘图

动作 — 移动光标, 或者定义操作的范围

命令 — 直接执行的命令,
红色命令 进入编辑模式

操作 — 后面跟随表示操作范围的指令

extra — 特殊功能,
需要额外的输入

q· — 后跟字符参数

w,e,b命令

小写(b): `quux(foo, bar, baz);`
大写(B): `quux(foo, bar, baz);`

主要ex命令:
:w (保存), **:q** (退出), **:q!** (不保存退出)
:e f (打开文件 **f**),
:%s/x/y/g (**'y'** 全局替换 **'x'**),
:h (帮助 **in vim**), **:new** (新建文件 **in vim**),

其它重要命令:
CTRL-R: 重复 **(vim)**,
CTRL-F/-B: 上翻/下翻,
CTRL-E/-Y: 上滚/下滚,
CTRL-V: 块可视模式 **(vim only)**

可视模式:
漫游后对选中的区域执行操作 **(vim only)**

备注:
(1) 在 拷贝/粘贴/删除 命令前使用 **"x** (**x=a..z,***)
使用命令的寄存器(**'剪贴板'**)
(如: **"ay$** 拷贝剩余的行内容至寄存器 **'a'**)
(2) 命令前添加数字
多遍重复操作
(**e.g.: 2p, d2w, 5i, d4j**)
(3) 重复本字符在光标所在行执行操作
(**dd** = 删除本行, **>>** = 行首缩进)
(4) **ZZ** 保存退出, **ZQ** 不保存退出
(5) **zt:** 移动光标所在行至屏幕顶端,
zb: 底端, **zz:** 中间
(6) **gg:** 文首 **(vim only)**,
gf: 打开光标处的文件名 **(vim only)**

原图: **www.viemu.com** 翻译: *fdl (linuxsir)*

史上最全Vim快捷键键位图

练习作业

- 用vim新建一个test.c文件
- 将xxx文件内容插入到test.c文件中
- 将当前文件中xxx字符串全部替换为yyy字符串
- 将当前文件中10-20行的代码注释掉
- 将2-3行代码复制粘贴10次

find 与 grep

- find命令是一个无处不在命令，是linux中最有用的命令之一。find命令用于：在一个目录（及子目录）中搜索文件，你可以指定一些匹配条件，如按文件名、文件类型、用户甚至是时间戳查找文件。

- grep (global search regular expression(RE) and print out the line,全面搜索正则表达式并把行打印出来)是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。

find用法举例

- find path -option [-print] [-exec -ok command] {} \;
- $find ~ -name "*.txt" -print #在$HOME中查.txt文件并显示
- $find . -name "*.txt" -print
- $find . -name "[A-Z]*" -print #查以大写字母开头的文件
- $find /etc -name "host*" -print #查以host开头的文件
- $find . -name "[a-z][a-z][0–9][0–9].txt" -print #查以两个小写字母和两个数字开头的txt文件

find用法举例

- $find . -perm 755 -print
- $find . -perm -007 -exec ls -l {} \; #查所有用户都可读写执行的文件同-perm 777
- $find . -type d -print
- $find . ! -type d -print
- $find . -type l -print
- $find . -size +1000000c -print #查长度大于1Mb的文件
- $find . -size 100c -print # 查长度为100c的文件
- $find . -size +10 -print #查长度超过10块的文件（1块=512字节）

grep用法举例

- grep 'test' d* 显示所有以d开头的文件中包含test的行。
- grep 'test' aa bb cc 显示在aa，bb，cc文件中匹配test的行。
- sudo grep -v “#" /etc/host.conf
- ls -l | grep '^a' 通过管道过滤ls -l输出的内容，只显示以a开头的行。

grep正则表达式元字符

- ^ 锚定行的开始 如：'^grep'匹配所有以grep开头的行。
- $ 锚定行的结束 如：'grep$'匹配所有以grep结尾的行。
- . 匹配一个非换行符的字符 如：'gr.p'匹配gr后接一个任意字符，然后是p。
- * 匹配零个或多个先前字符 如：' *grep'匹配所有一个或多个空格后紧跟grep的行。 .*一起用代表任意字符。
- [] 匹配一个指定范围内的字符，如'[Gg]rep'匹配Grep和grep。
- [^] 匹配一个不在指定范围内的字符，如：’[^A-FH-Z]rep'匹配不包含A-F和H-Z的一个字母开头，紧跟rep的行。
- .....

grep的选项

- -b 在每一行前面加上其所在的块号，根据上下文定位磁盘块时可能会用到
- -c 显示匹配到的行的数目，而不是显示行的内容
- -h 不显示文件名
- -i 比较字符时忽略大小写的区别
- -l（小写的字母L） 只列出匹配行所在文件的文件名（每个文件名只列一次），文件名之间用换行符分隔
- -n 在每一行前面加上它在文件中的相对行号
- -r 对目录下递归查询所有子目录
- -v 反向查找，只显示不匹配的行
- -w 把表达式作为词来查找，就好像它被\<和\>夹着那样。只适用于grep（并非所有版本的grep都支持这一功能，譬如，SCO UNIX就不支持）
- ….

练习作业

- 通过find找出/etc目录中所有*.conf；通过grep将*.conf文件中包含“ubuntu”的行输出到一个文本文件

Git

- Git是目前世界上最先进的分布式版本控制系统（没有之一）
- Subversion/svn
- Concurrent Versions System/cvs

Git的诞生

- 在2002年以前，Linux内核源代码文件是通过diff的方式发给Linus，然后由Linus本人通过手工方式合并代码！
- Linus坚定地反对CVS和SVN，这些集中式的版本控制系统不但速度慢，而且必须联网才能使用。有一些商用的版本控制系统，虽然比CVS、SVN好用，但那是付费的，和Linux的开源精神不符。
- 2002年Linus选择了一个商业的版本控制系统BitKeeper，BitKeeper的东家BitMover公司出于人道主义精神，授权Linux社区免费使用这个版本控制系统。
- 2005年BitMover公司要收回Linux社区的免费使用权。Linus花了两周时间自己用C写了一个分布式版本控制系统，这就是Git！一个月之内，Linux系统的源码已经由Git管理了！
- 2008年，GitHub网站上线了，它为开源项目免费提供Git存储。
- 2016年，11 Years After Git, BitKeeper Is Open-Sourced

Git的诞生

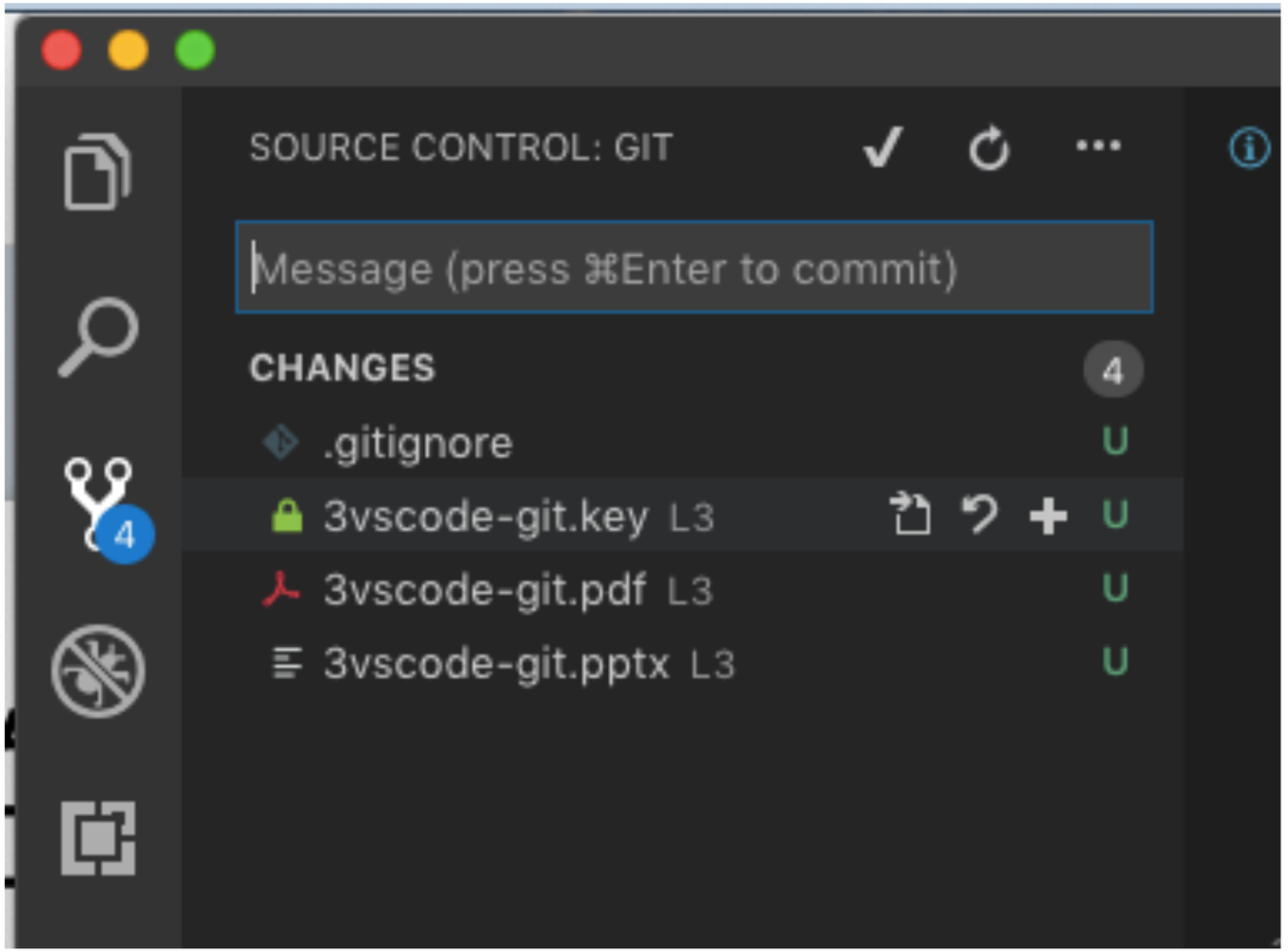
在Linux上安装Git并创建版本库

- sudo apt install git
- git init # 在一个新建的目录下创建版本库
- git clone https://github.com/mengning/linuxstart.git

git的常用命令

- git add [FILES] # 把文件添加到版本库
- git commit -m "wrote a commit log infro” # 把文件提交到仓库
- git log / git reflog
- git status
- git diff [FILES]
- git reset --hard HEAD^ # 回滚到指定版本，上一个版本就是HEAD^，上上一个版本就是HEAD^^
- git pull
- git push

Visual Studio Code & Git



练习作业

- 在github.com上新建一个版本库
- git clone
- 添加文件到版本库并push到github上

Regular Expression

- 正则表达式是对字符串操作的一种逻辑公式。
- 正则表达式的应用范围非常之广泛，最初是由Unix普及开来的，后来在广泛运用于Scala 、PHP、C# 、Java、C++ 、Objective-c、Perl 、Swift、VBScript 、Javascript、Ruby 以及Python等等。
- 学习正则表达式，实际上是在学习一种十分灵活的逻辑思维，通过简单快速的方法达到对于字符串的控制。
- **正则表达式是程序员手中一把威力无比强大的武器！**

为什么使用正则表达式?

- 测试字符串内的模式。例如，可以测试输入字符串，以查看字符串内是否出现电话号码模式或信用卡号码模式。这称为数据验证。

- 替换文本。可以使用正则表达式来识别文档中的特定文本，完全删除该文本或者用其他文本替换它。

- 基于模式匹配从字符串中提取子字符串。可以查找文档内或输入域内特定的文本。

- 例如，您可能需要搜索整个网站，删除过时的材料，以及替换某些 HTML 格式标记。在这种情况下，可以使用正则表达式来确定在每个文件中是否出现该材料或该 HTML 格式标记。此过程将受影响的文件列表缩小到包含需要删除或更改的材料的那些文件。然后可以使用正则表达式来删除过时的材料。最后，可以使用正则表达式来搜索和替换标记。

作业

- 如何熟练掌握软件开发中的常用工具?
  - Vim训练游戏?
  - 正则表达式练习游戏?
  - find、grep、git……

参考资料

- https://github.com/mengning/linuxstart/
- https://mp.weixin.qq.com/s/4sKrDYzAFv8dAAP_tiYCMw
- https://www.w3cschool.cn/zhengzebiaodashi/