# 命令行环境

**结束进程**

[kill](https://www.man7.org/linux/man-pages/man1/kill.1.html) 命令：kill -TERM <PID>

**暂停和后台执行进程**

SIGSTOP 进程暂停。

在终端中，键入 Ctrl-Z 会让 shell 发送 SIGTSTP 信号，SIGTSTP 是 Terminal Stop 的缩写（即 terminal 版本的 SIGSTOP）

[fg](https://www.man7.org/linux/man-pages/man1/fg.1p.html) 或 [bg](http://man7.org/linux/man-pages/man1/bg.1p.html) 命令恢复暂停的工作

[jobs](http://man7.org/linux/man-pages/man1/jobs.1p.html) 命令会列出当前终端会话中尚未完成的全部任务

百分号 + 任务编号（jobs 会打印任务编号）来选取该任务

选择最近的一个任务，可以使用 $! 这一特殊参数

让已经在运行的进程转到后台运行，您可以键入 Ctrl-Z ，然后紧接着再输入 bg。注意，后台的进程仍然是您的终端进程的子进程，一旦您关闭终端（会发送另外一个信号 SIGHUP），这些后台的进程也会终止。为了防止这种情况发生，您可以使用 [nohup](https://www.man7.org/linux/man-pages/man1/nohup.1.html)（一个用来忽略 SIGHUP 的封装）来运行程序。针对已经运行的程序，可以使用 disown 。

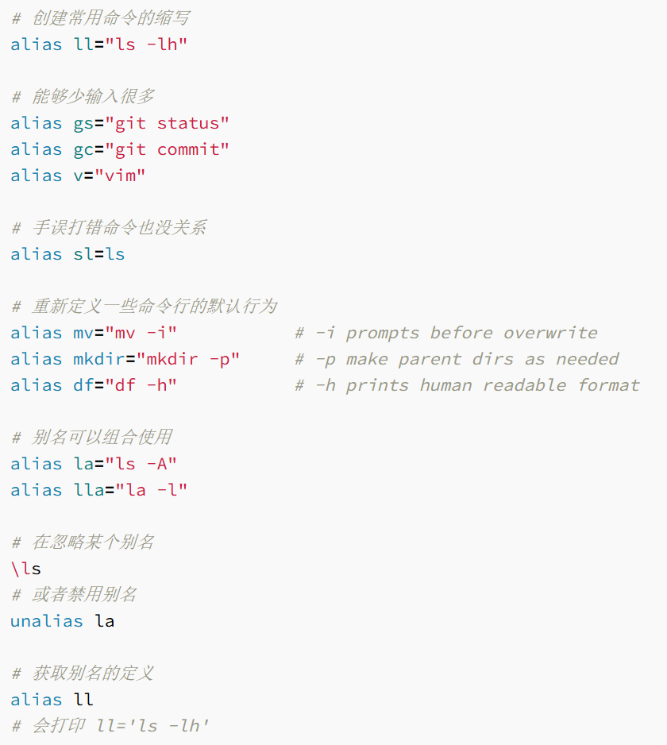
# 终端多路复用

像 [tmux](https://www.man7.org/linux/man-pages/man1/tmux.1.html) 这类的终端多路复用器可以允许我们基于面板和标签分割出多个终端窗口，这样您便可以同时与多个 shell 会话进行交互。不仅如此，终端多路复用使我们可以分离当前终端会话并在将来重新连接。

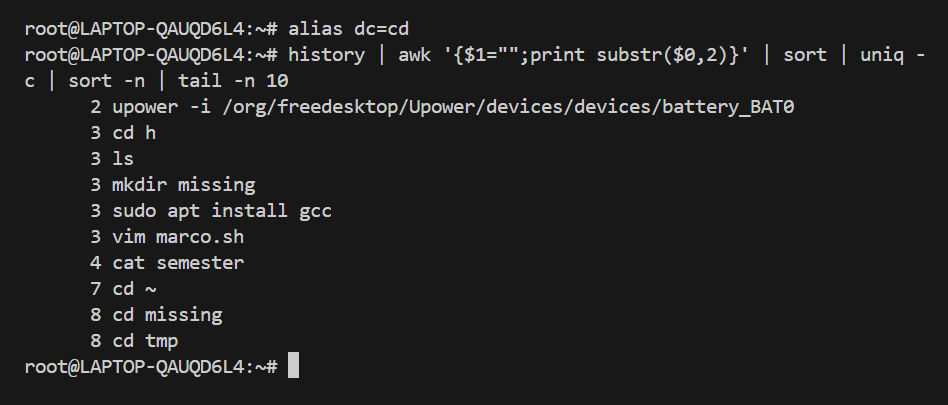
* **会话** - 每个会话都是一个独立的工作区，其中包含一个或多个窗口
  + tmux 开始一个新的会话
  + tmux new -s NAME 以指定名称开始一个新的会话
  + tmux ls 列出当前所有会话
  + 在 tmux 中输入 <C-b> d ，将当前会话分离
  + tmux a 重新连接最后一个会话。您也可以通过 -t 来指定具体的会话
* **窗口** - 相当于编辑器或是浏览器中的标签页，从视觉上将一个会话分割为多个部分
  + <C-b> c 创建一个新的窗口，使用 <C-d> 关闭
  + <C-b> N 跳转到第 N 个窗口，注意每个窗口都是有编号的
  + <C-b> p 切换到前一个窗口
  + <C-b> n 切换到下一个窗口
  + <C-b> , 重命名当前窗口
  + <C-b> w 列出当前所有窗口
* **面板** - 像 vim 中的分屏一样，面板使我们可以在一个屏幕里显示多个 shell
  + <C-b> " 水平分割
  + <C-b> % 垂直分割
  + <C-b> <方向> 切换到指定方向的面板，<方向> 指的是键盘上的方向键
  + <C-b> z 切换当前面板的缩放
  + <C-b> [ 开始往回卷动屏幕。您可以按下空格键来开始选择，回车键复制选中的部分
  + <C-b> <空格> 在不同的面板排布间切换

扩展阅读： [这里](https://www.hamvocke.com/blog/a-quick-and-easy-guide-to-tmux/) 是一份 tmux 快速入门教程， [而这一篇](http://linuxcommand.org/lc3_adv_termmux.php) 文章则更加详细，它包含了 screen 命令。您也许想要掌握 [screen](https://www.man7.org/linux/man-pages/man1/screen.1.html) 命令，因为在大多数 UNIX 系统中都默认安装有该程序。

# 别名



值得注意的是，在默认情况下 shell 并不会保存别名。为了让别名持续生效，您需要将配置放进 shell 的启动文件里



# 配置文件（Dotfiles）



# 版本控制(Git)

# Git 的命令行接口

## 基础

* git help <command>: 获取 git 命令的帮助信息
* git init: 创建一个新的 git 仓库，其数据会存放在一个名为 .git 的目录下
* git status: 显示当前的仓库状态
* git add <filename>: 添加文件到暂存区
* git commit: 创建一个新的提交
  + 如何编写 [良好的提交信息](https://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html)!
  + 为何要 [编写良好的提交信息](https://chris.beams.io/posts/git-commit/)
* git log: 显示历史日志
* git log --all --graph --decorate: 可视化历史记录（有向无环图）
* git diff <filename>: 显示与暂存区文件的差异
* git diff <revision> <filename>: 显示某个文件两个版本之间的差异
* git checkout <revision>: 更新 HEAD 和目前的分支

## 分支和合并

* git branch: 显示分支
* git branch <name>: 创建分支
* git checkout -b <name>: 创建分支并切换到该分支
  + 相当于 git branch <name>; git checkout <name>
* git merge <revision>: 合并到当前分支
* git mergetool: 使用工具来处理合并冲突
* git rebase: 将一系列补丁变基（rebase）为新的基线

## 远端操作

* git remote: 列出远端
* git remote add <name> <url>: 添加一个远端
* git push <remote> <local branch>:<remote branch>: 将对象传送至远端并更新远端引用
* git branch --set-upstream-to=<remote>/<remote branch>: 创建本地和远端分支的关联关系
* git fetch: 从远端获取对象/索引
* git pull: 相当于 git fetch; git merge
* git clone: 从远端下载仓库

## 撤销

* git commit --amend: 编辑提交的内容或信息
* git reset HEAD <file>: 恢复暂存的文件
* git checkout -- <file>: 丢弃修改
* git restore: git2.32 版本后取代 git reset 进行许多撤销操作

# Git 高级操作

* git config: Git 是一个 [高度可定制的](https://git-scm.com/docs/git-config) 工具
* git clone --depth=1: 浅克隆（shallow clone），不包括完整的版本历史信息
* git add -p: 交互式暂存
* git rebase -i: 交互式变基
* git blame: 查看最后修改某行的人
* git stash: 暂时移除工作目录下的修改内容
* git bisect: 通过二分查找搜索历史记录
* .gitignore: [指定](https://git-scm.com/docs/gitignore) 故意不追踪的文件

# 调试及性能分析

Python 的调试器是 [pdb](https://docs.python.org/3/library/pdb.html).

下面对 pdb 支持的命令进行简单的介绍：

* **l**(ist) - 显示当前行附近的 11 行或继续执行之前的显示；
* **s**(tep) - 执行当前行，并在第一个可能的地方停止；
* **n**(ext) - 继续执行直到当前函数的下一条语句或者 return 语句；
* **b**(reak) - 设置断点（基于传入的参数）；
* **p**(rint) - 在当前上下文对表达式求值并打印结果。还有一个命令是 **pp** ，它使用 [pprint](https://docs.python.org/3/library/pprint.html) 打印；
* **r**(eturn) - 继续执行直到当前函数返回；
* **q**(uit) - 退出调试器。