文件导航

列出目录内容

```
$ ls          # 查看目录
$ ls -l       # 格式化查看目录
$ ls -la      # 格式化查看目录包括隐藏文件
```

切换目录

```
$ cd [path]   # 切换目录
$ cd ..       # 切换到上级目录
$ pwd         # 显示当前所在目录
```

创建和删除目录境

```
$ mkdir
$ rm -r       # 删除目录
$ rm -f       # 强制删除
```

文件操作

```
$ cp          # 复制文件
$ mv          # 重命名或移动文件
$ touch file  # 创建文件
$ cat [file]  # 输出文件内容
$ cat > [file]  # 将标准输入写入文件
$ cat >> [file] # 追加标准输入到文件
$ tail -f [file] # 动态输出文件最后内容
```

网络

网络诊断和查询

```
$ ping [host]
$ dig [domain]      # 获取域的DNS
$ dig -x [host]     # 反向查询主机
$ route -n          # 显示实际的路由表
$ iptables -L       # 检查你的 iptable 规则
$ whois [domain]    # 查询域名的whois
$ netstat -a        # 列出所有的端口
```

网络下载

```
$ wget [file]       # 下载文件
$ wget -r [url]     # 从url递归下载文件
$ curl [url]        # 输出网页内容
```

SSH

```
$ ssh user@host     # 通过 user 连接到主机
$ ssh -p [port] user@host  # 指定端口连接
$ ssh -D user@host  # 连接并使用绑定端口
```

进程

```
$ ps          # 显示当前活跃进程
$ ps -aux     # 显示细节信息
$ kill [pid]  # 杀死进程
$ killall proc  # 杀死所有名称为proc的进程
```

系统信息

资源信息

```
$ cat /proc/cpuinfo   # 显示CPU信息
$ cat /proc/meminfo.  # 显示内存信息
$ free                # 显示内存和交换内存使用情况
$ du                  # 显示目录空间使用情况
$ du -sh              # 用GB显示可读的大小
$ df                  # 显示磁盘的使用情况
```

其他信息

```
$ date        # 日期时间
$ uptime      # 正常运行时间
$ whoami      # 你是谁
$ w           # 当前谁在线
$ uname -a    # 内核配置信息
```

压缩

```
$ tar -cf [file.tar] [files]  # 将 files 压缩进 file.tar 中
$ tar -xf [file.tar]          # 解压缩到当前目录
```

其他

权限

```
$ chmod [rights] [file]  # 修改文件权限
```

查找

```
$ grep '[pattern]' [files]  # 按模式搜索文件
$ grep -r '[pattern]' dir   # 在目录中递归搜索
$ locate [file]             # 查找文件的所有实例
$ whereis [app]             # 可执行文件可能在哪里
```

编译和部署

编译可执行文件

```
$ go build -o=/tmp/foo .                # 编译当前目录的包
$ go build -o=/tmp/foo ./cmd/foo        # 编译 ./cmd/foo 目录的包
```

编译缓存

```
$ go env GOCACHE                        # 检查你的编译缓存存放目录
$ go build -a -o=/tmp/foo .             # 强制重编译所有包
$ go clean  -cache                      # 清除缓存
```

编译过程

```
$ go list -deps . | sort -u             # 列出用于编译可执行文件的所有包
$ go build -a -x -o=/tmp/foo .          # 重建所有内容并显示运行的命令
```

交叉编译

```
$ GOOS=linux GOARCH=amd64 go build -o=/tmp/linux_amd64/foo .
$ GOOS=windows GOARCH=amd64 go build -o=/tmp/windows_amd64/foo.exe .
$ go tool dist list                     # 列出所有支持的操作系统和 CPU架构
```

使用编译器和链接器 flags

```
$ go tool compile -help                                 # 查看编译器可用的 flag
$ go build -gcflags="-m -m" -o=/tmp/foo .               # 打印有关优化决策的信息
$ go build -gcflags="all=-m" -o=/tmp/foo .              # 打印包括依赖的优化决策信息
$ go build -gcflags="all=-N -l" -o=/tmp/foo .           # 禁用优化和内联
$ go tool link -help                                    # 查看链接器可用的 flag
$ go build -ldflags="-X main.version=1.2.3" -o=/tmp/foo . # 增加版本信息
$ go build -ldflags="-s -w" -o=/tmp/foo .               # 从二进制文件中删除调试信息
$ CGO_ENABLE=0 GOOS=linux go build -a -ldflags '-extldflags "-static"' .    # 使二进制文件尽可能静态
```

问题诊断和优化

运行和比较基准测试

```
$ go test -bench=. ./...                #运行所有测试和基准测试
$ go test -run=^$ -bench=. ./...        # 只运行所有的基准测试
$ go test -run=^$ -bench=^BenchmarkFoo$ ./...  # 只运行 BenchmarkFoo 基准测试
$ go test -bench=. -benchmem ./...      # 强制将内存分配统计信息包含在输出中
$ go test -bench=. -benchtime=5s ./...  # 运行每个基准测试至少5秒钟
$ go test -bench=. -benchtime=500x ./...  # 保证每个基准测试精确进行500次迭代
$ go test -bench=. -count=3 ./...       # 重复每个基准测试3次
$ go test -bench=. -cpu=1,4,8 ./....    #将GOMAXPROCS设置为1、4和8运行基准测试
```

比较基准测试的变化

```
$ cd /tmp  # 安装 benchcmp 工具
$ GO111MODULE=on go get golang.org/x/tools/cmd/benchcmp
$ go test -run=^$ -bench=. -benchmem ./... > /tmp/old.txt
# 进行一些修改、优化
$ go test -run=^$ -bench=. -benchmem ./... > /tmp/new.txt
$ benchcmp /tmp/old.txt /tmp/new.txt
```

Profiling and Tracing

运行和比较基准测试

```
$ go test -run="$ -bench="BenchmarkFoo$ -cpuprofile=/tmp/cpuprofile.out .
$ go test -run="$ -bench="BenchmarkFoo$ -memprofile=/tmp/memprofile.out .
$ go test -run="$ -bench="BenchmarkFoo$ -blockprofile=/tmp/blockprofile.out .
$ go test -run="$ -bench="BenchmarkFoo$ -mutexprofile=/tmp/mutexprofile.out .
$ go test -run="$ -bench="BenchmarkFoo$ -o=/tmp/foo.test -cpuprofile=/tmp/cpuprofile.out .
$ go tool pprof -http=:5000 /tmp/cpuprofile.out   # 在浏览器中审查
$ go tool pprof --nodefraction=0.1 -http=:5000 /tmp/cpuprofile.out      # 忽略小于 10% 的节点
```

Tracing 生成

```
$ go test -run="$ -bench="BenchmarkFoo$ -trace=/tmp/trace.out .
$ go tool trace /tmp/trace.out                                  # 目前只在 Chrome/Chromium 可用
```

竞态条件检查

```
$ go build -race -o=/tmp/foo .                  # 别用于生产环境
$ GORACE="log_path=/tmp/race" /tmp/foo          # 输出到文件而不是标准错误
```

依赖管理（Module）

项目依赖更新

```
$ go list -m -u github.com/alecthomas/chroma    # 检查该库是否有新版本
$ go list -m -u all                             # 更新项目所有依赖
```

依赖升级或降级

```
$ go get github.com/foo/bar@latest      # 最新版本
$ go get github.com/foo/bar@v1.2.3      # 特定版本 v1.2.3
$ go get github.com/foo/bar@7e0369f     # 到特定提交
```

运行所有包的全部测试检验不兼容性

```
$ go mod tidy
$ go test all
```

使用依赖的本地版本

```
$ go mod edit -replace=github.com/alexedwards/argon2id=/home/alex/code/argon2id  # 创建 replace 规则
$ go mod edit -dropreplace=github.com/alexedwards/argon2id                       # 删除 replace 规则
```

其他工具

升级代码到 Go 新版本

```
$ go fix ./...
```

报告 Bug

```
$ go bug                # 会打开浏览器，定位 Go 代码仓库的 issue 页面
```