Go's Tooling Cheat Sheet

Installing Tooling

Easiest way

```
$ go get -u golang.org/x/tools/...
```

Alternative way

```
$ cd /tmp          # To avoid to install into a local project as a module
$ GO111MODULE=on go get golang.org/x/tools/cmd/stress
```

Viewing Environment Information

```
$ go env
$ go env GOPATH GOOS GOARCH
$ go help environment
```

Development

Running Code

```
$ go run .          # Run the package in the current directory
$ go run ./cmd/foo  # Run the package in the ./cmd/foo directory
```

Fetching Dependencies

```
$ go get github.com/foo/bar@v1.2.3
$ go get github.com/foo/bar@8e1b8d3
$ go list -m all                 # Show all the dependencies
$ go mod why -m golang.org/x/sys  # Why is that a dependency?
$ go clean -modcache             # clear module cache
```

Refactoring Code

`gofmt -d -w -r 'foo -> Foo' .`	Replace foo by Foo
`gofmt -w -r 'strings.Replace(a, b, -1) -> strings.ReplaceAll(a, b)' .`	

Viewing Go Documentation

```
$ go doc strings                # View simplified documentation for the strings package
$ go doc -all strings           # View full documentation for the strings package
$ go doc strings.Replace        # View documentation for the strings.Replace function
$ go doc sql.DB                 # View documentation for the database/sql.DB type
$ go doc sql.DB.Query           # View documentation for the database/sql.DB.Query method
$ go doc -src strings.Replace   # View the source code for the strings.Replace function
```

Testing

Running Tests

```
$ go test .                           # Run all tests in the current directory
$ go test ./...                       # Run all tests in the current directory and sub-directories
$ go test ./foo/bar                   # Run all tests in the ./foo/bar directory
$ go test -race ./...                 # Testing with race detector
$ go test -count=1 ./...              # Bypass the test cache when running tests
$ go clean -testcache                 # Delete all cached test results
$ go test -v -run=^TestFooBar$ .      # Run the test with the exact name TestFooBar
$ go test -v -run=^TestFoo .          # Run tests whose names start with TestFoo
$ go test -v -run=^TestFooBar$/^Baz$ . # Run the Baz subtest of the TestFooBar test only
$ go test -short ./...                # handy flag - skip long running tests
$ go test -failfast ./...             # handy flag - don't run further tests after a failure.
```

Profiling Test Coverage

```
$ go test -cover ./...
$ go test -coverprofile=/tmp/profile.out ./...                  # coverage profile for browser
$ go tool cover -html=/tmp/profile.out
$ go test -covermode=count -coverprofile=/tmp/profile.out ./... # coverage with frequency shown
$ go tool cover -html=/tmp/profile.out
$ go test -coverprofile=/tmp/profile.out ./...                  # coverage in CLI without any browser
$ go tool cover -func=/tmp/profile.out
```

Stress Testing

```
$ go test -run=^TestFooBar$ -count=500 .
$ go test -c -o=/tmp/foo.test .                                 # using stress tool
$ stress -p=4 /tmp/foo.test -test.run=^TestFooBar$
```

Testing all dependencies

```
$ go test all
```

Pre-Commit Checks

Formatting code

```
$ gofmt -w -s -d foo.go  # Format the foo.go file
$ gofmt -w -s -d .       # Recursively format everything
$ go fmt ./...           # alternative formatting tool
```

Performing Static Analysis with vet

```
$ go vet foo.go                 # Vet the foo.go file
$ go vet .                      # Vet all files in the current directory
$ go vet ./...                  # Vet all files in the current directory and sub-directories
$ go vet ./foo/bar              # Vet all files in the ./foo/bar directory
$ go vet -composites=false ./... # Disable some analyzers
```

Experimental analyzers

```
$ cd /tmp
$ GO111MODULE=on go get golang.org/x/tools/go/analysis/passes/nilness/cmd/nilness
$ GO111MODULE=on go get golang.org/x/tools/go/analysis/passes/shadow/cmd/shadow
$ go vet -vettool=$(which nilness) ./...
```

Disable vet checks before running any tests

```
$ go test -vet=off ./...
```

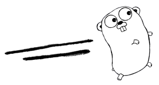
Linting Code

```
$ cd /tmp          # installing the linter
$ GO111MODULE=on go get golang.org/x/lint/golint
$ golint foo.go    # Lint the foo.go file
$ golint .         # Lint all files in the current directory
$ golint ./...     # Lint all files in the current directory and sub-directories
$ golint ./foo/bar # Lint all files in the ./foo/bar directory
```

Tidying and verifying Your dependencies

```
$ go mod tidy      # prune any unused dependencies
$ go mod verify    # check the dependencies' hashes
```

Go's Tooling Cheat Sheet



Build and Deployment

Building an Executable

```
$ go build -o=/tmp/foo .            # Compile the package in the current directory
$ go build -o=/tmp/foo ./cmd/foo    # Compile the package in the ./cmd/foo directory
```

Build cache

```
$ go env GOCACHE                    # Check where your build cache is
$ go build -a -o=/tmp/foo .         # Force all packages to be rebuilt
$ go clean -cache                   # Remove everything from the build cache
```

Cross-Compilation

```
$ 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                 # list of all supported OS/architectures
```

Using Compiler and Linker Flags

```
$ go tool compile -help                        # complete list of available compiler flags
$ go build -gcflags="-m -m" -o=/tmp/foo .      # print optimization decisions
$ go build -gcflags="all=-m" -o=/tmp/foo .     # optimization decisions for dependencies too
$ go build -gcflags="all=-N -l" -o=/tmp/foo .  # disable optimizations and inlining
$ go tool link -help                           # list of available linker flags
$ go build -ldflags="-X main.version=1.2.3" -o=/tmp/foo . # add a version number
$ go build -ldflags="-s -w" -o=/tmp/foo .      # strip debug information from the binary
$ CGO_ENABLED=0 GOOS=linux go build -a -ldflags '-extldflags "-static"' . # make the binary as static as possible
```

Diagnosing Problems and Making Optimizations

Running and Comparing Benchmarks

```
$ go test -bench=. ./...                        # Run all benchmarks and tests
$ go test -run=^$ -bench=. ./...                # Run all benchmarks (and no tests)
$ go test -run=^$ -bench=^BenchmarkFoo$ ./...   # Run only the BenchmarkFoo benchmark (and no tests)
$ go test -bench=. -benchmem ./...              # Forces the output of memory allocation statistics
$ go test -bench=. -benchtime=5s ./...          # Run each benchmark test for at least 5 seconds
$ go test -bench=. -benchtime=500x ./...        # Run each benchmark test for exactly 500 iterations
$ go test -bench=. -count=3 ./...               # Repeat each benchmark test 3 times over
$ go test -bench=. -cpu=1,4,8 ./...             # Run benchmarks with GOMAXPROCS set to 1, 4 and 8
```

Comparing changes between benchmarks

```
$ cd /tmp # Installing
$ GO111MODULE=on go get golang.org/x/tools/cmd/benchcmp
$ go test -run=^$ -bench=. -benchmem ./... > /tmp/old.txt
# make changes
$ go test -run=^$ -bench=. -benchmem ./... > /tmp/new.txt
$ benchcmp /tmp/old.txt /tmp/new.txt
```

Profiling and Tracing

Running and comparing benchmarks

```
$ 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                    # inspecting a profile in a browser
$ go tool pprof --nodefraction=0.1 -http=:5000 /tmp/cpuprofile.out  # ignore nodes smaller than 10%
```

Trace generation

```
$ go test -run=^$ -bench=^BenchmarkFoo$ -trace=/tmp/trace.out .
$ go tool trace /tmp/trace.out            # Works only on Chrome / Chromium at the moment
```

Checking for Race Conditions

```
$ go build -race -o=/tmp/foo .            # not for production
$ GORACE="log_path=/tmp/race" /tmp/foo    # specify the output to a fiel instead of stderr
```

Managing Dependencies

```
$ go list -m -u github.com/alecthomas/chroma     # check for updates to all dependencies
```

Upgrade (or downgrade) a dependency

```
$ go get github.com/foo/bar@latest
$ go get github.com/foo/bar@v1.2.3
$ go get github.com/foo/bar@7e0369f
```

Run the tests for all packages to help check for incompatibilities

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

Use a local version of a dependency

```
$ go mod edit -replace=github.com/alexedwards/argon2id=/home/alex/code/argon2id    # create the replace rule
$ go mod edit -dropreplace=github.com/alexedwards/argon2id                         # remove the replace rule
```

Upgrading the code to a New Go Release

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

Reporting Bugs

```
$ go bug         # create a new Github issue for Go's standard library
```