Unit : 1 Introduction

- History
- Philosophy
- Sweet spots

Introduction

- *Go* also known as *Golang* is an open source, compiled and statically typed programming language developed by Google
- The key people behind the creation of Go are Rob Pike, Ken Thompson and Robert Griesemer
- Go was made publicly available in November 2009
- Go is a general-purpose programming language with a simple syntax and is backed by a robust standard library

Introduction

- ➢ One of the key areas where Go shines is the creation of highly available and scalable web apps
- ➢ Go can also be used to create command-line applications, desktop apps and even mobile applications

Advantages of Go

- Why would you choose Go as your server side programming language
  - when there are tons of other languages such as python, ruby, nodejs... that do the same job

Here are some of the pros

- **Simple syntax :** The syntax is simple and concise and the language is not bloated with unnecessary features
    - This makes it easy to write code that is readable and maintainable
- **Easy to write concurrent programs :** Concurrency is an inherent part of the language
    - As a result, writing multithreaded programs is a piece of cake

Advantages of Go

- **Compiled language :** Go is a compiled language
  - The source code is compiled to a native binary
  - This is missing in interpreted languages such as JavaScript used in nodejs
- **Fast compilation**
  - The Go compiler is pretty amazing and it has been designed to be fast right from the beginning
- **Static linking :** The Go compiler supports static linking

Advantages of Go

- ➢ The entire Go project can be statically linked into one big fat binary and it can be deployed in cloud servers easily without worrying about dependencies
- ➢ **Go Tooling :** Tooling deserves a special mention in Go
- ➢ Go comes bundled with powerful tools that help developers write better code

Advantages of Go

Few of the commonly used tools are:

- ➢ **gofmt** - gofmt is used to automatically format go source code
  - ○ It uses tabs for indentation and blanks for alignment
- ➢ **vet** - vet analyses the go source code and reports possible suspicious code
  - ○ Everything reported by vet is not a genuine problem but it has the capability to catch errors that are not reported by the compiler such as incorrect format specifiers when using Printf
- ➢ **golint** - golint is used to identify styling issues in the code

Advantages of Go

- **Garbage collection :** Go uses garbage collection and hence memory management is pretty much taken care automatically
  - The developer doesn't need to worry about managing memory
  - This also helps to write concurrent programs easily
- **Simple language specification**
  - The language spec is pretty simple
  - The entire spec fits in a page and you can even use it to write your own compiler :)

Advantages of Go

- **Open Source**
  - Last but not least, Go is an open source project
  - You can participate and contribute to the Go project

Popular products built using Go

- The following are some of the popular products built using Go
  - Google developed Kubernetes using Go
  - Docker, the world famous containerization platform is developed using Go
  - Dropbox has migrated its performance critical components from Python to Go
  - Infoblox's next generation networking products are developed using Go

History

- Go was designed at Google in 2007 to improve programming productivity in an era of multicore, networked machines and large codebases
- The designers wanted to address criticism of other languages in use at Google, but keep their useful characteristics:
    - static typing and run-time efficiency (like C),
    - readability and usability (like Python or JavaScript),
    - high-performance networking and multiprocessing

History

- ➢ The designers were primarily motivated by their shared dislike of C++
- ➢ Go was publicly announced in November 2009, and version 1.0 was released in March 2012
- ➢ Go is widely used in production at Google and in many other organizations and open-source projects
- ➢ In November 2016, the Go and Go Mono fonts were released by type designers Charles Bigelow and Kris Holmes specifically for use by the Go project

History

- ➢ Go font is a humanist sans-serif which resembles Lucida Grande and Go Mono font is monospaced
- ➢ Each of the fonts adhere to the WGL4 character set and were designed to be legible with a large x-height and distinct letterforms
- ➢ Both Go and Go Mono adhere to the DIN 1450 standard by having a slashed zero, lowercase l with a tail, and an uppercase I with serifs
- ➢ In April 2018, the original logo was replaced with a stylized GO slanting right with trailing streamlines

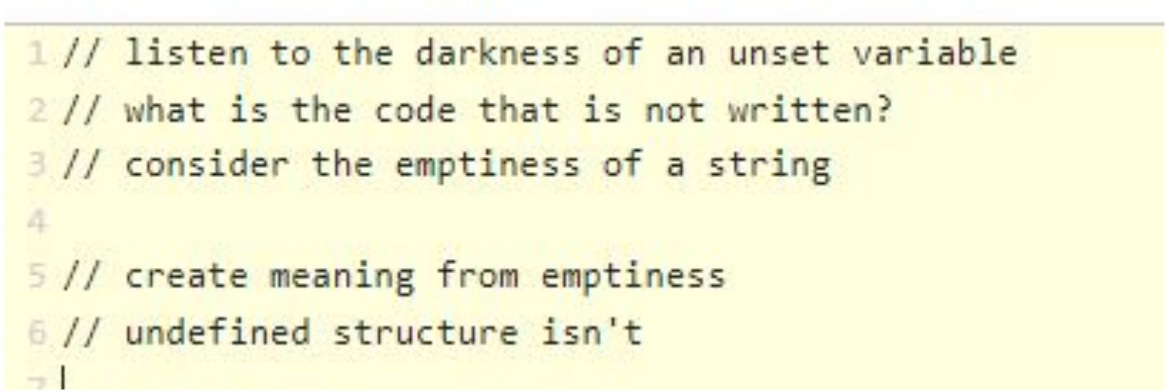
History

- ➢ However, the Gopher mascot remained the same
- ➢ In August 2018, the Go principal contributors published two "draft designs" for new and incompatible "Go 2" language features, generics and error handling, and asked Go users to submit feedback on them
- ➢ Lack of support for generic programming and the verbosity of error handling in Go 1.x had drawn considerable criticism

Mascot of Go programming language is a Gopher shown above

golang philosophy

- 1. The uninitialized variable - with default value (0,’’, nil)



```
// listen to the darkness of an unset variable
// what is the code that is not written?
// consider the emptiness of a string

// create meaning from emptiness
// undefined structure isn't
```

golang philosophy

- **2. The implications of a string**
- One of the most deceptively simple types in modern programming languages is the string
- In Go, there is a built-in string type with short, unsatisfying descriptive text
- Strings, bytes, runes and characters in Go explains that strings are a read-only slice of bytes (at runtime)
- Go source code is UTF-8, so string literals always contain UTF-8 text (except for byte-level escapes)

Golang philosophy

- Strings always trigger reflection on how memory management works in a language

Sweet Spots

- Many programming languages are “general purpose” languages, but most of them have a sweet spot if not a specific problem domain
- If you program for long enough and you’re willing to try new things, your favorite tool for specific jobs will likely get replaced
- The story on the server side in general has been a bit bumpier as of late
- In the 90’s virtually a lot of server side work was in C/C++

Sweet Spots

- In the 2000s a lot of projects moved, reluctantly, to Java. While Java’s restrictions and resource usage are not very well liked,
  - it was easier to build and deploy, more portable, and less error prone
  - The result was it was easier and faster to build correct systems.
- Java is abound with boilerplate and repetition
  - For e.g.: projects like Spring, AspectJ, and cglib

Sweet Spots

- ➢ Go imposes a strict file layout so that *all of your projects*, even if they are completely unrelated to each other, have to live in a specific global directory layout
- ➢ Deviations from this will break your code
  - ○ expect that a sizable portion of new Go developers' second or third projects are build tools to make using Go less painful

Setting Up a Development Environment

- Tour of Playground
- Installation and setup
- Create first application

Tour of Playground

- ➢ The Go Playground is a web service that runs on go.dev's servers
- ➢ The service receives a Go program, vets, compiles, links, and runs the program inside a sandbox, then returns the output
- ➢ If the program contains tests or examples and no main function, the service runs the tests
- ➢ Benchmarks will likely not be supported since the program runs in a sandboxed environment with limited resources

Tour of Playground

- There are limitations to the programs that can be run in the playground:
  - The playground can use most of the standard library, with some exceptions
  - The only communication a playground program has to the outside world is by writing to standard output and standard error
  - In the playground the time begins at 2009-11-10 23:00:00 UTC (determining the significance of this date is an exercise for the reader)

Tour of Playground

  - This makes it easier to cache programs by giving them deterministic output
  - There are also limits on execution time and on CPU and memory usage
- The playground uses the latest stable release of Go
- The playground service is used by more than just the official Go project

Tour of Playground

- ➢ Use a unique user agent in your requests (so we can identify you), and that your service is of benefit to the Go community
- ➢ Any requests for content removal should be directed to security@golang.org
- ➢ Please include the URL and the reason for the request

How to Install Go on Windows

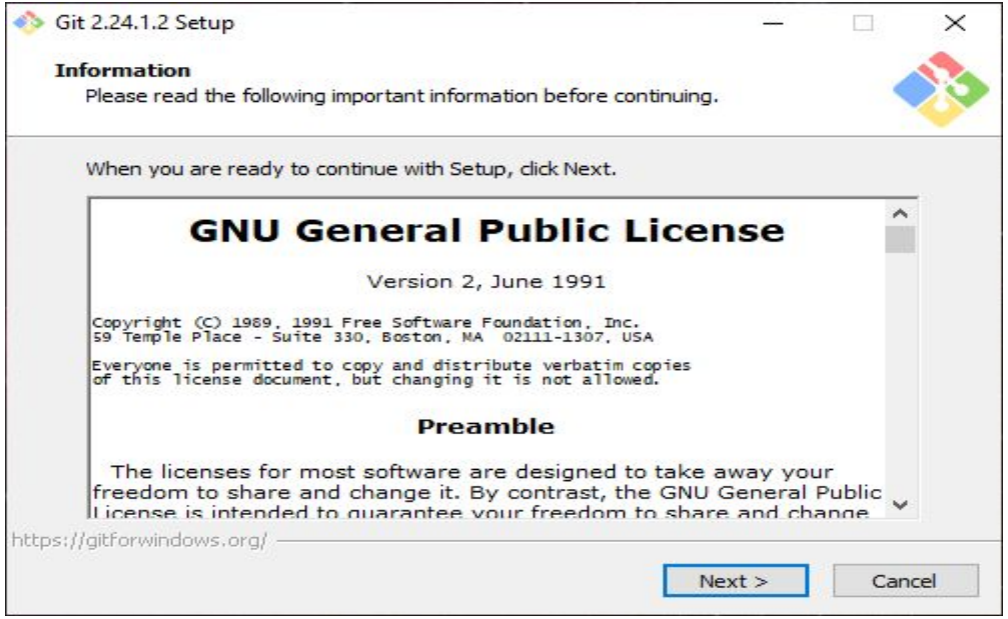
- ➢ **Prerequisites for GoLang**
- ➢ Need a version control software to manage go projects
- ➢ Here we will use git
- ➢ **Installing git**
- ➢ First install git
- ➢ If you have already installed git, you can skip this step

Download git installer

- ➢ Go to git-scm.com and click on downloads
- ➢ Select your platform
- ➢ Here we are using windows
- ➢ The download will start

Install git on Windows

- ➢ After download, launch the **.exe** file
- ➢ The installation wizard will show up and license will be shown
- ➢ Click next



Information

Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

GNU General Public License

Version 2, June 1991

```
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA  02111-1307, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

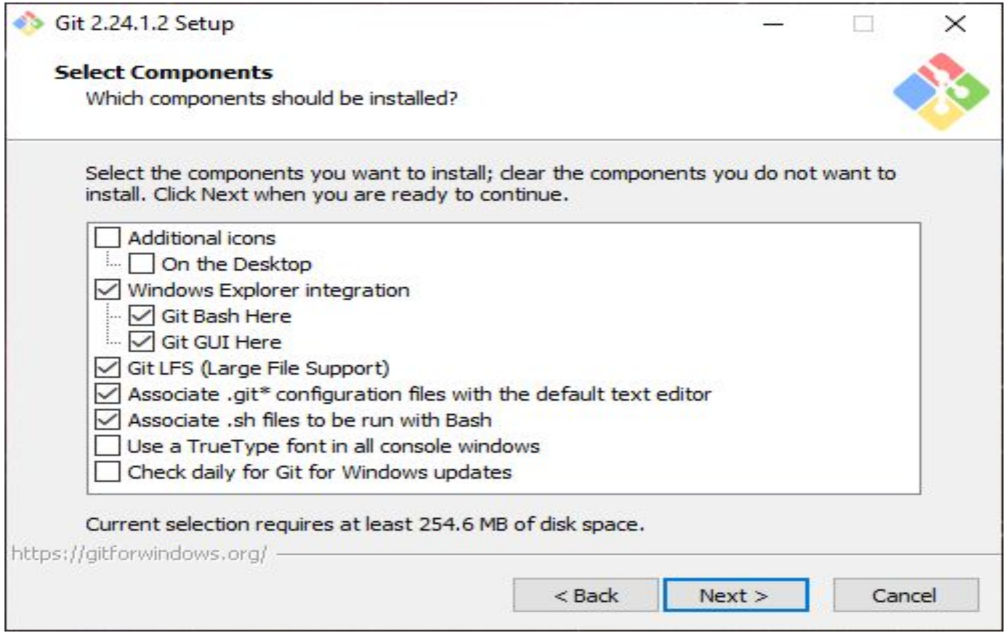
Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change

https://gitforwindows.org/

Install git on Windows

- ➢ Here you have to select the components to install
- ➢ Keep this as it is if you are a beginner and click next



Git 2.24.1.2 Setup

Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

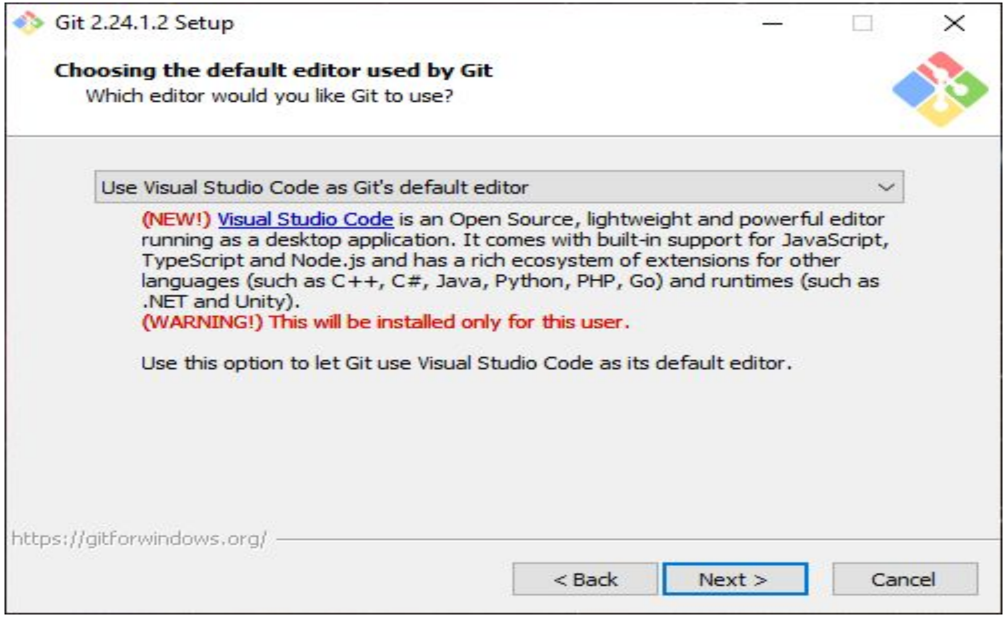
- ☐ Additional icons
  - ☐ On the Desktop
- ☑ Windows Explorer integration
  - ☑ Git Bash Here
  - ☑ Git GUI Here
- ☑ Git LFS (Large File Support)
- ☑ Associate .git* configuration files with the default text editor
- ☑ Associate .sh files to be run with Bash
- ☐ Use a TrueType font in all console windows
- ☐ Check daily for Git for Windows updates

Current selection requires at least 254.6 MB of disk space.

https://gitforwindows.org/

Install git on Windows

- ➢ Now you will have to select the text editor of your choice
- ➢ Here I have Visual Studio Code already installed so it is selected by default



Choosing the default editor used by Git
Which editor would you like Git to use?

Use Visual Studio Code as Git's default editor

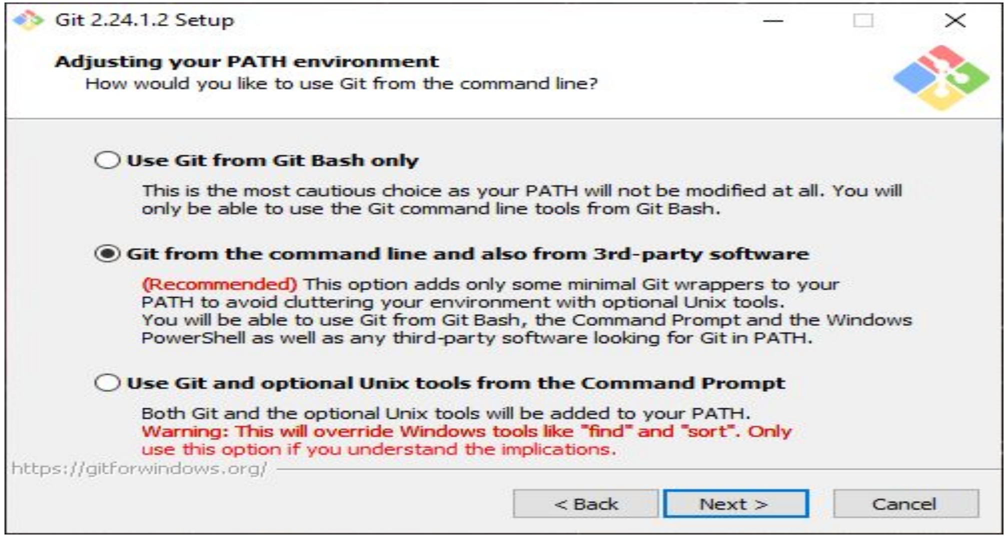
(NEW!) Visual Studio Code is an Open Source, lightweight and powerful editor running as a desktop application. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).
(WARNING!) This will be installed only for this user.

Use this option to let Git use Visual Studio Code as its default editor.

Install git on Windows

- ➢ Now you will have to select how you will use git from the command line
- ➢ Select the first option if you are beginning
- ➢ If you select the second option, you will be able to use git from any command-line software like CMD, Powershell to name a few
- ➢ Click next after you have selected the option accordingly

Install git on Windows



Git 2.24.1.2 Setup

Adjusting your PATH environment

How would you like to use Git from the command line?

( ) **Use Git from Git Bash only**

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

[✓ **Git from the command line and also from 3rd-party software**]

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools. You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

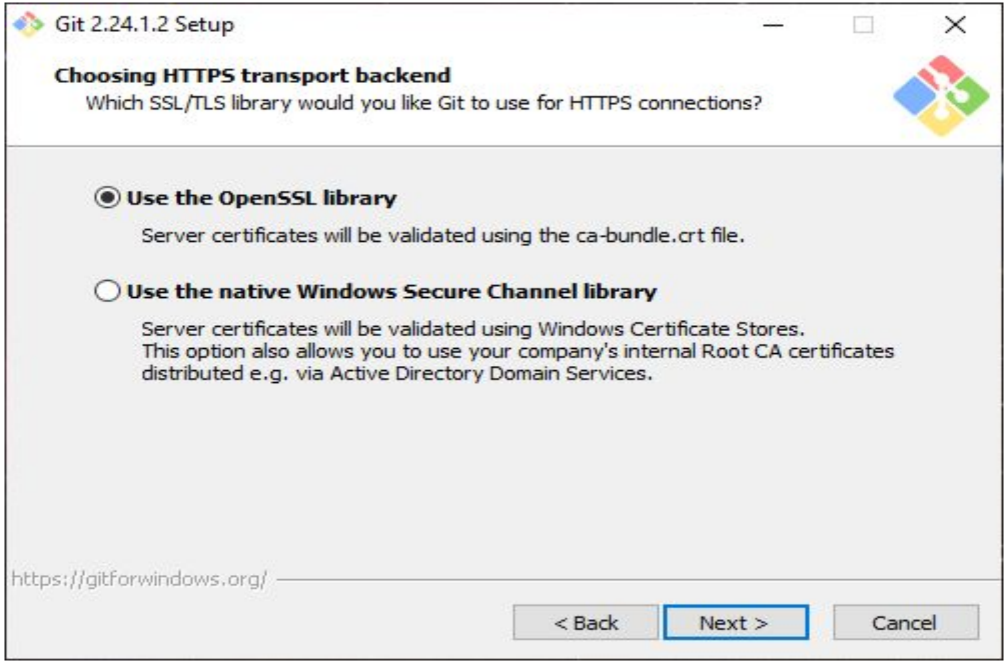
( ) **Use Git and optional Unix tools from the Command Prompt**

Both Git and the optional Unix tools will be added to your PATH. Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

- ➢ In the next prompt, you have to select the https library that will be used when creating https connections using git
- ➢ OpenSSL library is selected by default

Install git on Windows

➢ Keep it as it is. Click next



Git 2.24.1.2 Setup

Choosing HTTPS transport backend

Which SSL/TLS library would you like Git to use for HTTPS connections?

● **Use the OpenSSL library**

Server certificates will be validated using the ca-bundle.crt file.

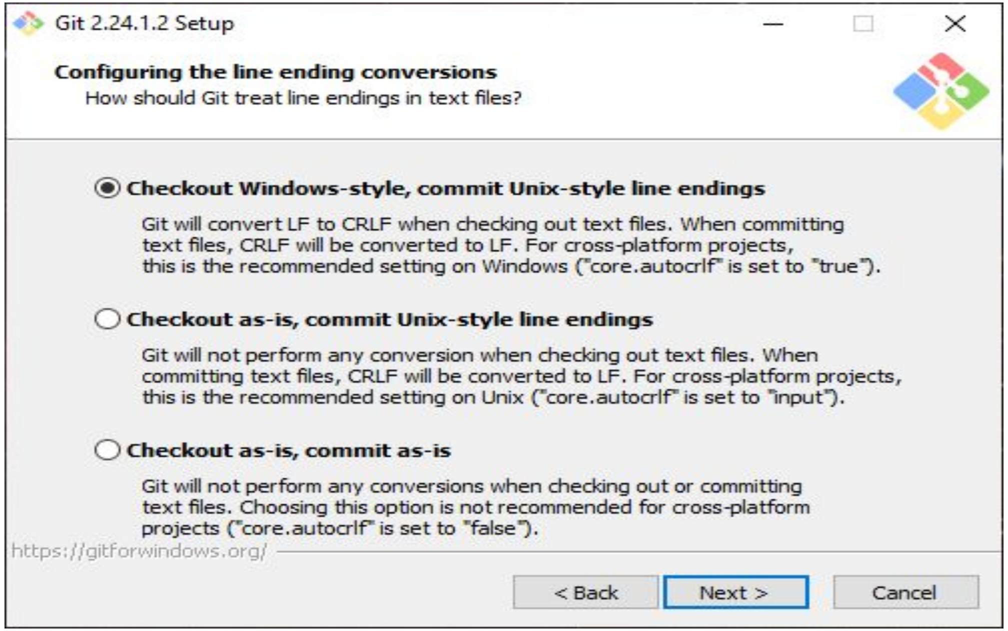
○ **Use the native Windows Secure Channel library**

Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates
distributed e.g. via Active Directory Domain Services.

https://gitforwindows.org/

Install git on Windows

- ➢ Here we select line ending options
- ➢ The default option will suffice
- ➢ Click next



Git 2.24.1.2 Setup

Configuring the line ending conversions

How should Git treat line endings in text files?

[✓ **Checkout Windows-style, commit Unix-style line endings**]

Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

Checkout as-is, commit Unix-style line endings

Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

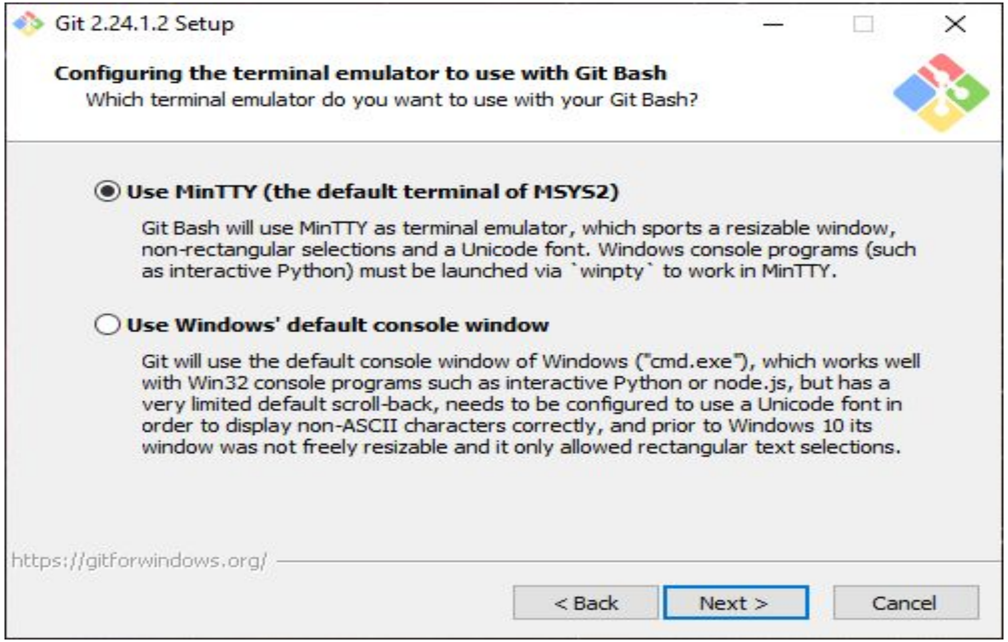
Checkout as-is, commit as-is

Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

https://gitforwindows.org/

Install git on Windows

- ➢ Now select the terminal emulator that will be used
- ➢ MinTTY is selected by default
- ➢ Click next


Git 2.24.1.2 Setup

Configuring the terminal emulator to use with Git Bash

Which terminal emulator do you want to use with your Git Bash?

[✓ **Use MinTTY (the default terminal of MSYS2)**]

Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

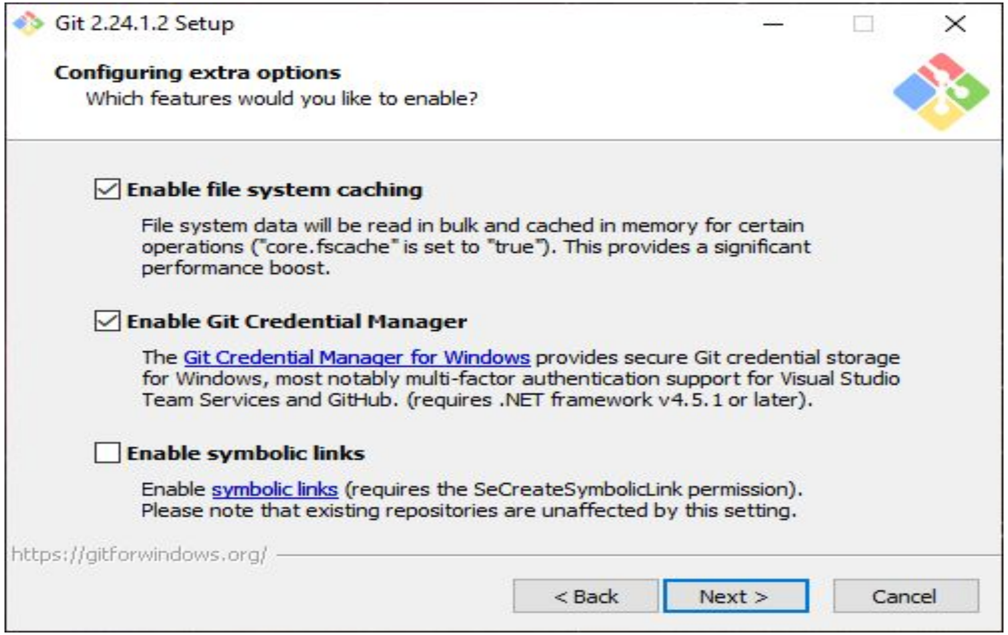
Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

https://gitforwindows.org/

Install git on Windows

- ➢ Then the extra options are shown
- ➢ Just click next and keep it as it is



Git 2.24.1.2 Setup

Configuring extra options
Which features would you like to enable?

[✓] **Enable file system caching**

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

[✓] **Enable Git Credential Manager**

The Git Credential Manager for Windows provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or later).

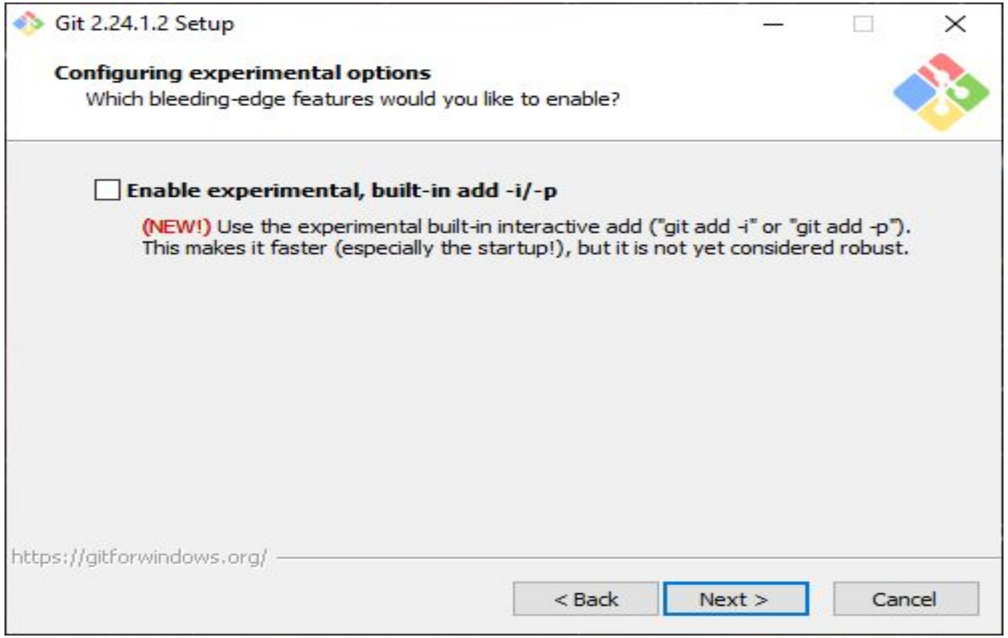
[ ] **Enable symbolic links**

Enable symbolic links (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

https://gitforwindows.org/

Install git on Windows

- Experimental options are not selected by default
- Can simply move forward and click next


Git 2.24.1.2 Setup

Configuring experimental options
Which bleeding-edge features would you like to enable?

[ ] **Enable experimental, built-in add -i/-p**
(NEW!) Use the experimental built-in interactive add ("git add -i" or "git add -p"). This makes it faster (especially the startup!), but it is not yet considered robust.

Install git on Windows

- ➢ It will install git on your Windows
- ➢ After the installation has ended, click finish
- ➢ You have successfully installed git
- ➢ Next, we set up a **git user**

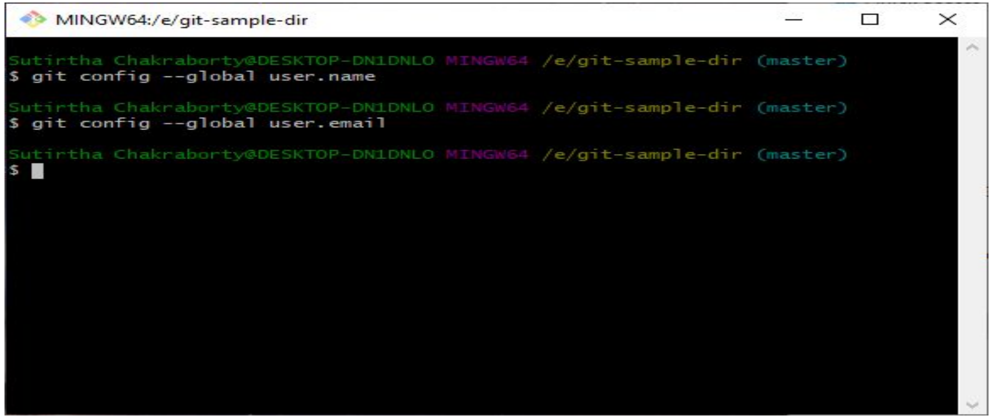
Set up git user

- After installing git you have to **set up a user**
- First, launch **git bash** from the start menu or desktop icon
- **Note that, in order to set up the user you need to be inside a git directory**
- Create an empty directory
- Then run the command below from inside it

```
git init
```

Set up git user

- ➢ You will notice **(master)** is written in that git-sample-directory I created
- ➢ That means it is a git directory



MINGW64:/e/git-sample-dir

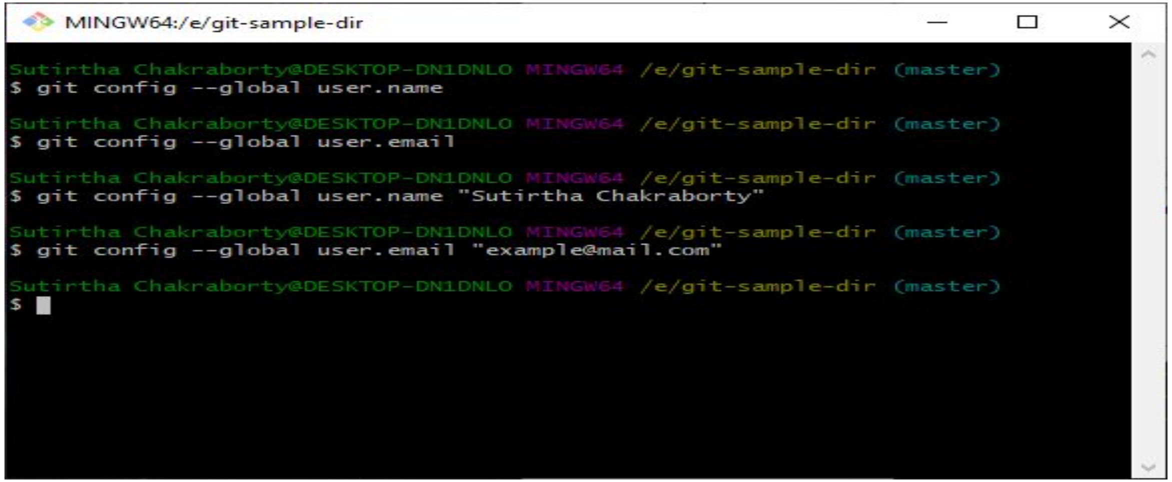
```
Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$ git config --global user.name

Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$ git config --global user.email

Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$
```

- ➢ In the image shown above, you can see the **user is not set**
- ➢ So, let’s set the git user

Set up git user

➢ The commands to set up user are:

```
git config --global user.name "your username"
git config --global user.email "your email"
```



MINGW64:/e/git-sample-dir

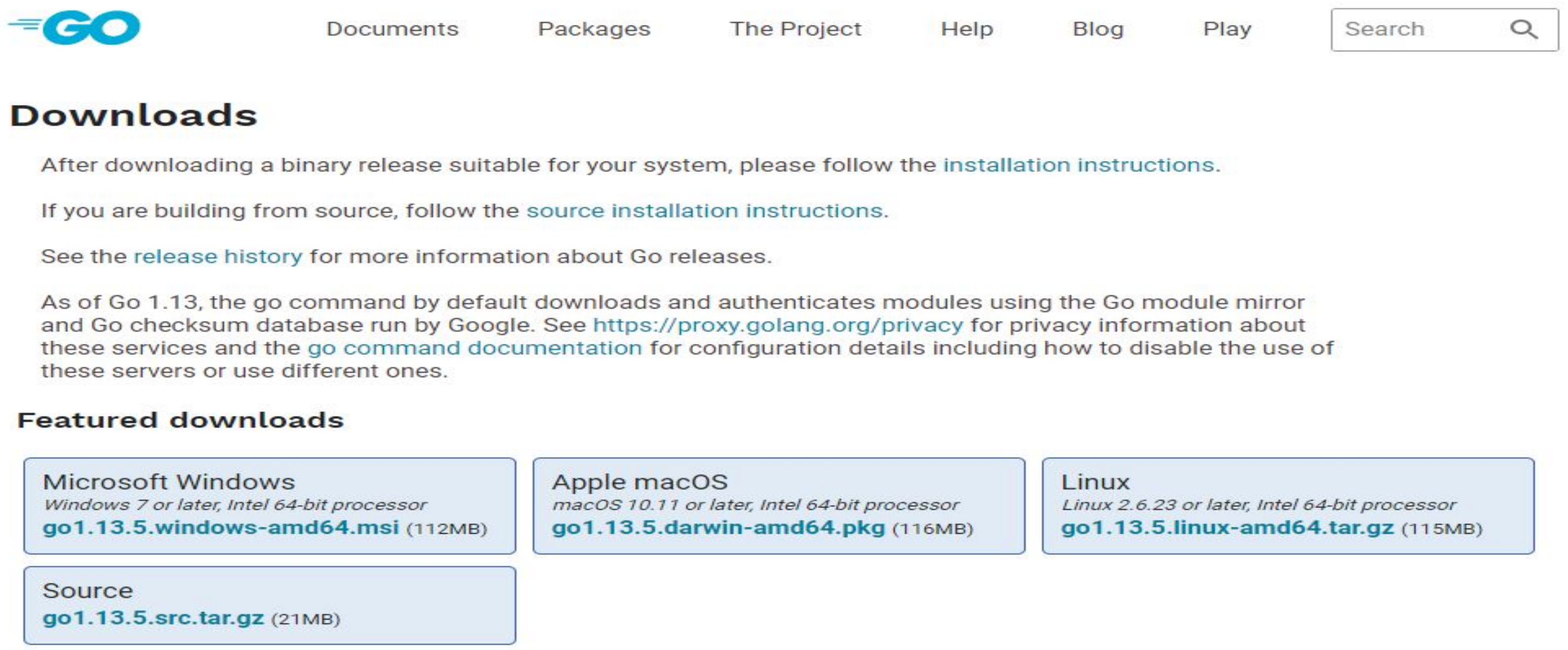
```
Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$ git config --global user.name

Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$ git config --global user.email

Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$ git config --global user.name "Sutirtha Chakraborty"

Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$ git config --global user.email "example@mail.com"

Sutirtha Chakraborty@DESKTOP-DN1DNLO MINGW64 /e/git-sample-dir (master)
$ 
```

➢ Done! You have successfully set yourself up as a git user

Installing GoLang

- **Download go for Windows**
- First, go to golang.org and click on “**Download Go**“
- It will lead to the download page
- There you will have to select the **msi installer for windows**

Installing GoLang



Downloads

After downloading a binary release suitable for your system, please follow the installation instructions.

If you are building from source, follow the source installation instructions.

See the release history for more information about Go releases.

As of Go 1.13, the go command by default downloads and authenticates modules using the Go module mirror and Go checksum database run by Google. See https://proxy.golang.org/privacy for privacy information about these services and the go command documentation for configuration details including how to disable the use of these servers or use different ones.

Featured downloads

Microsoft Windows
Windows 7 or later, Intel 64-bit processor
go1.13.5.windows-amd64.msi (112MB)

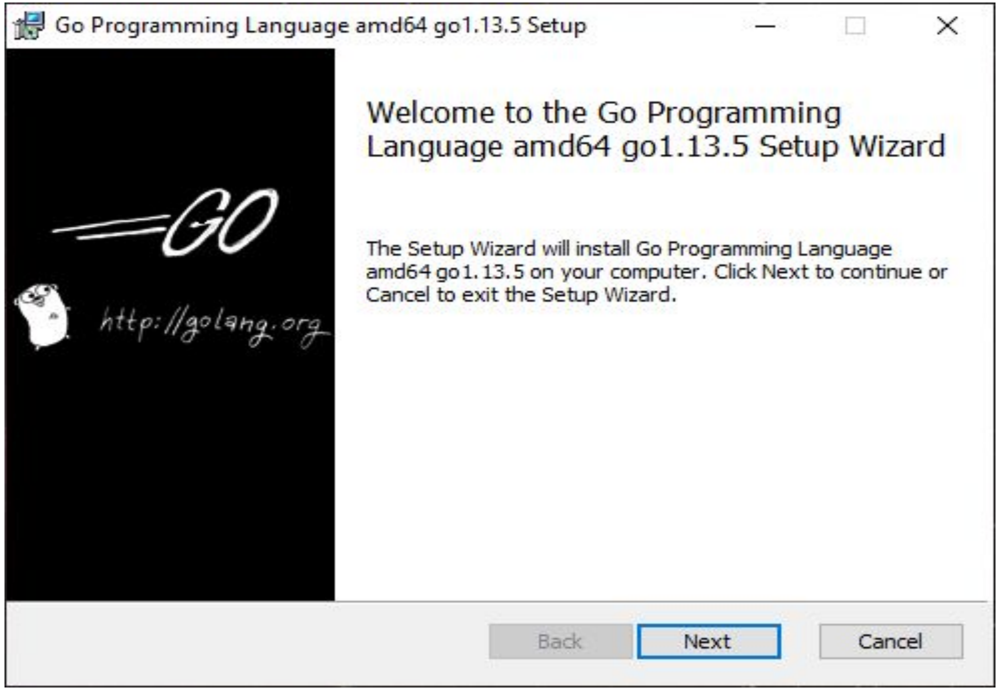
Apple macOS
macOS 10.11 or later, Intel 64-bit processor
go1.13.5.darwin-amd64.pkg (116MB)

Linux
Linux 2.6.23 or later, Intel 64-bit processor
go1.13.5.linux-amd64.tar.gz (115MB)

Source
go1.13.5.src.tar.gz (21MB)

Install go from .msi file

- ➢ After launching the installer (.msi file) the installer will begin
- ➢ Click next



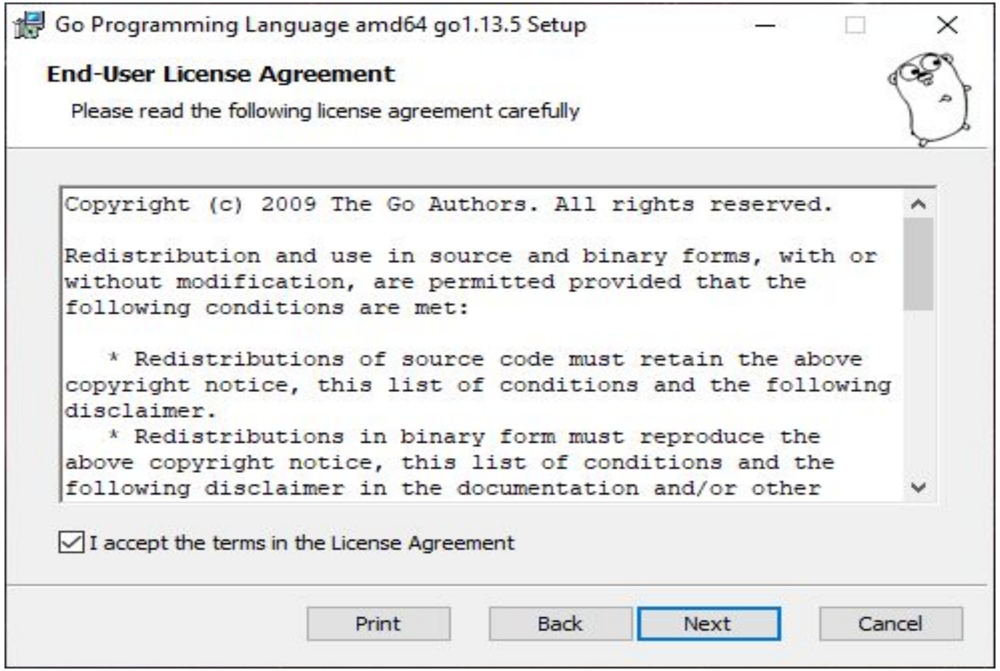
Go Programming Language amd64 go1.13.5 Setup

Welcome to the Go Programming Language amd64 go1.13.5 Setup Wizard

The Setup Wizard will install Go Programming Language amd64 go1.13.5 on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Install go from .msi file

- ➢ First, accept the license agreement
- ➢ Then click next



Go Programming Language amd64 go1.13.5 Setup

End-User License Agreement

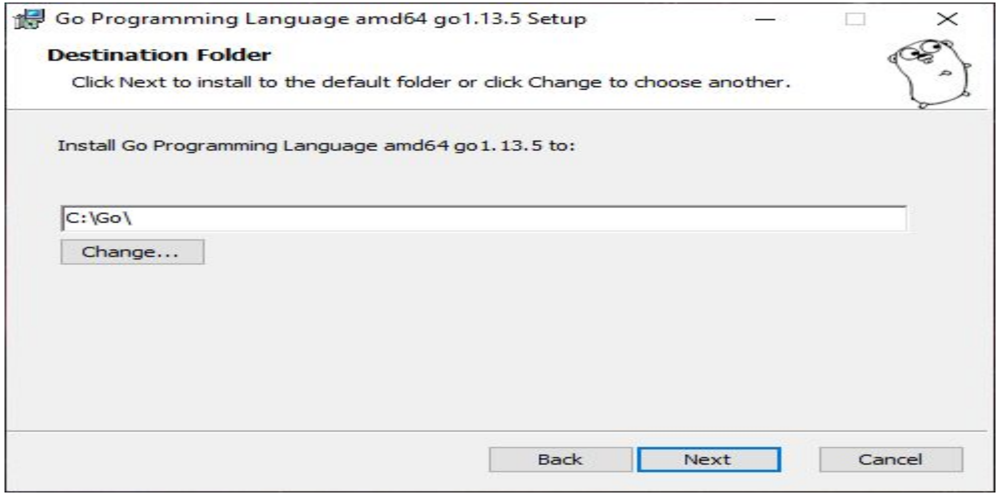
Please read the following license agreement carefully

```
Copyright (c) 2009 The Go Authors. All rights reserved.

Redistribution and use in source and binary forms, with or
without modification, are permitted provided that the
following conditions are met:

   * Redistributions of source code must retain the above
copyright notice, this list of conditions and the following
disclaimer.
   * Redistributions in binary form must reproduce the
above copyright notice, this list of conditions and the
following disclaimer in the documentation and/or other
```

☑ I accept the terms in the License Agreement

Install go from .msi file

- ➢ The destination is already set to a value
- ➢ If you would like to change it, just click change
- ➢ But keeping it to default value is just fine


Go Programming Language amd64 go1.13.5 Setup

Destination Folder

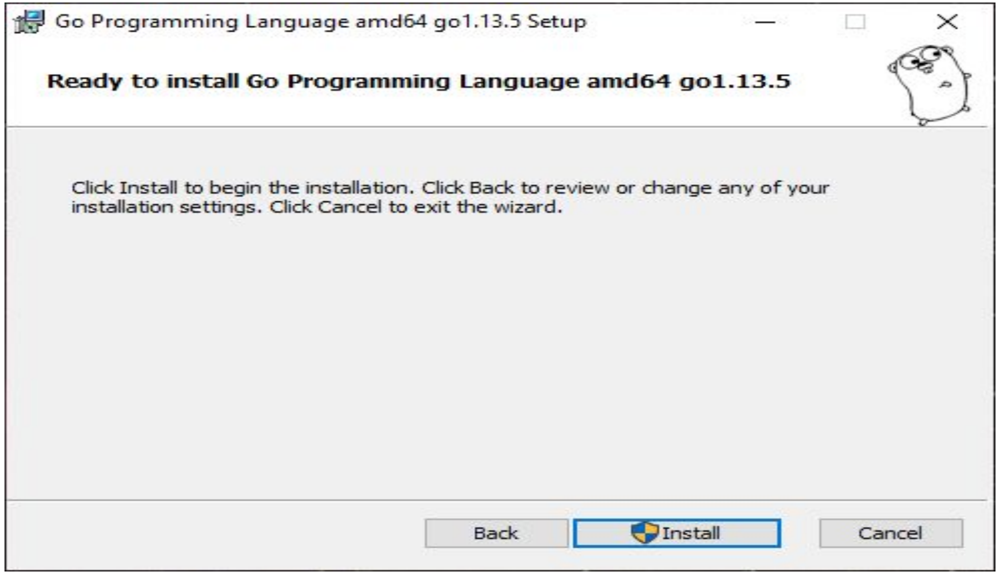
Click Next to install to the default folder or click Change to choose another.

Install Go Programming Language amd64 go1.13.5 to:

C:\Go\

Install go from .msi file

- ➢ Click Install to begin the installation
- ➢ Windows will ask for administrator permission
- ➢ Just accept it

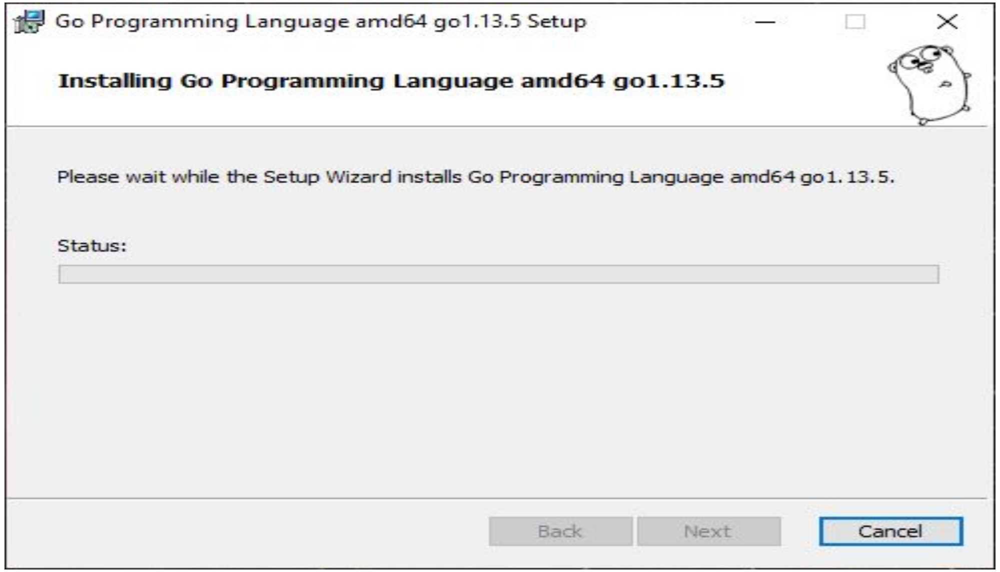

Go Programming Language amd64 go1.13.5 Setup

Ready to install Go Programming Language amd64 go1.13.5

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Install go from .msi file

➢ The installation has begun



Go Programming Language amd64 go1.13.5 Setup

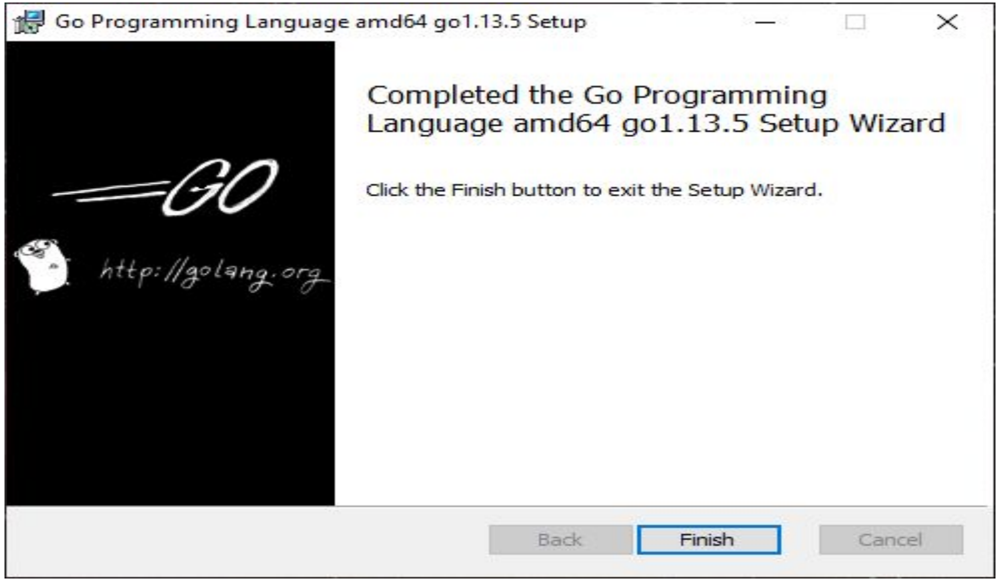
Installing Go Programming Language amd64 go1.13.5

Please wait while the Setup Wizard installs Go Programming Language amd64 go1. 13.5.

Status:

Install go from .msi file

- ➢ After it has completed installing, click **Finish** to complete the setup


Go Programming Language amd64 go1.13.5 Setup

Completed the Go Programming Language amd64 go1.13.5 Setup Wizard

Click the Finish button to exit the Setup Wizard.

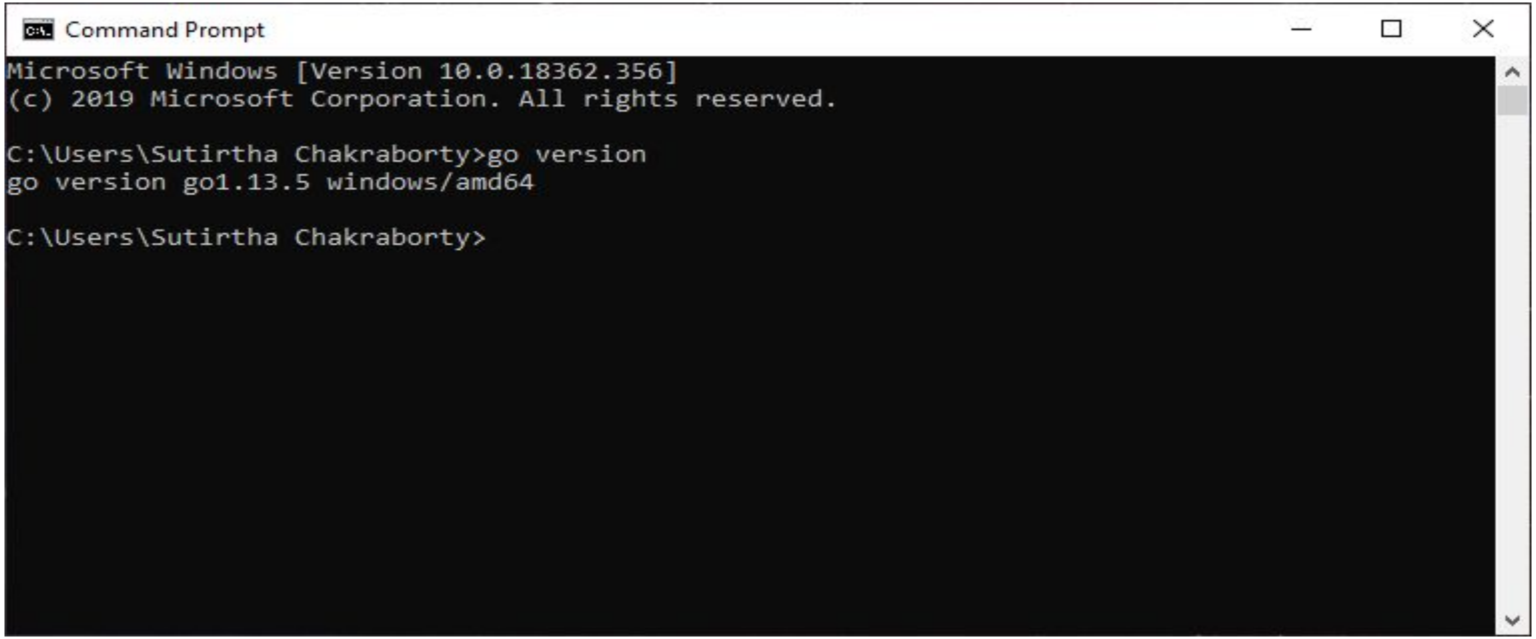
Install go from .msi file

- ➢ Golang has been installed successfully
- ➢ Now we need to check if it has been set up properly
- ➢ We can check that from the command prompt or any other terminal
- ➢ Here we will use CMD

Check GoLang version installed

➢ Enter the command below to check installed GoLang version:

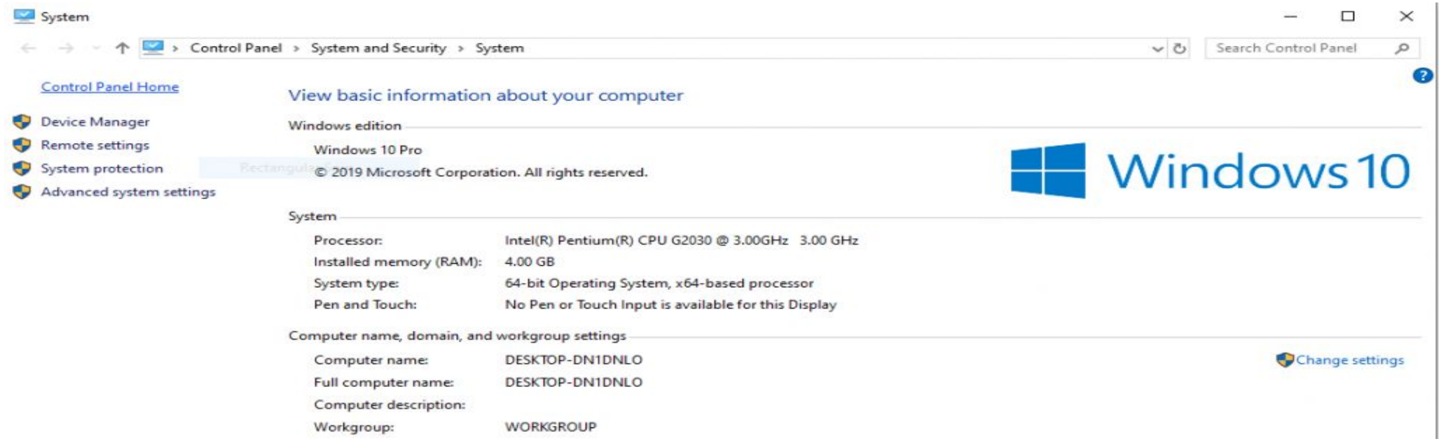
```
go version
```



```
Microsoft Windows [Version 10.0.18362.356]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Sutirtha Chakraborty>go version
go version go1.13.5 windows/amd64

C:\Users\Sutirtha Chakraborty>
```

Check if GOPATH is set up properly

- ➢ Now we need to check if GOPATH is set up properly
- ➢ To check that open **Control Panel**
- ➢ Then go to **System and Security** and click on **System**
- ➢ A window will open as shown below



View basic information about your computer

Windows edition

Windows 10 Pro

© 2019 Microsoft Corporation. All rights reserved.

System

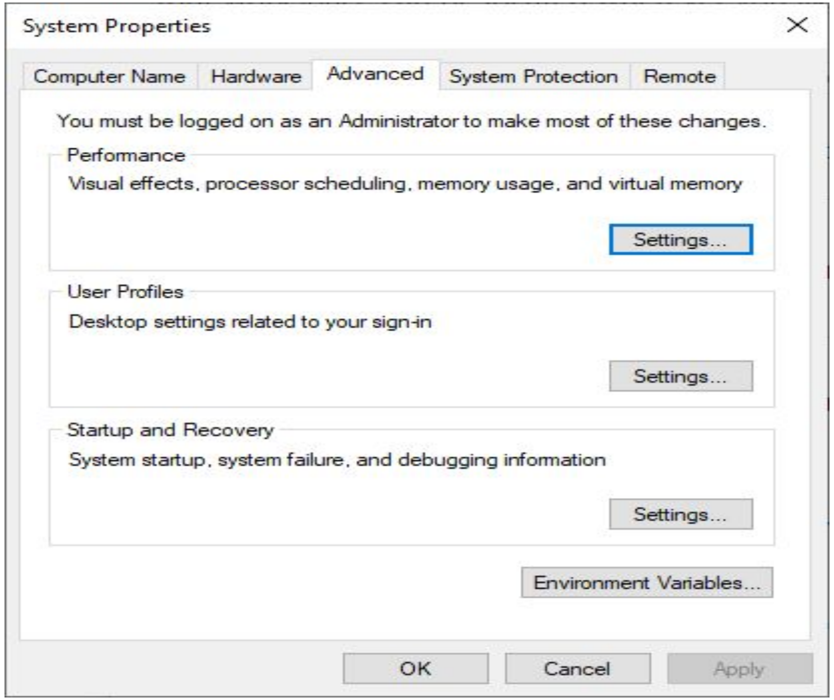
Processor:	Intel(R) Pentium(R) CPU G2030 @ 3.00GHz 3.00 GHz
Installed memory (RAM):	4.00 GB
System type:	64-bit Operating System, x64-based processor
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	DESKTOP-DN1DNLO
Full computer name:	DESKTOP-DN1DNLO
Computer description:	
Workgroup:	WORKGROUP

Check if GOPATH is set up properly

- Open **Advanced System Settings** from left sidebar


System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

User Profiles
Desktop settings related to your sign-in
Settings...

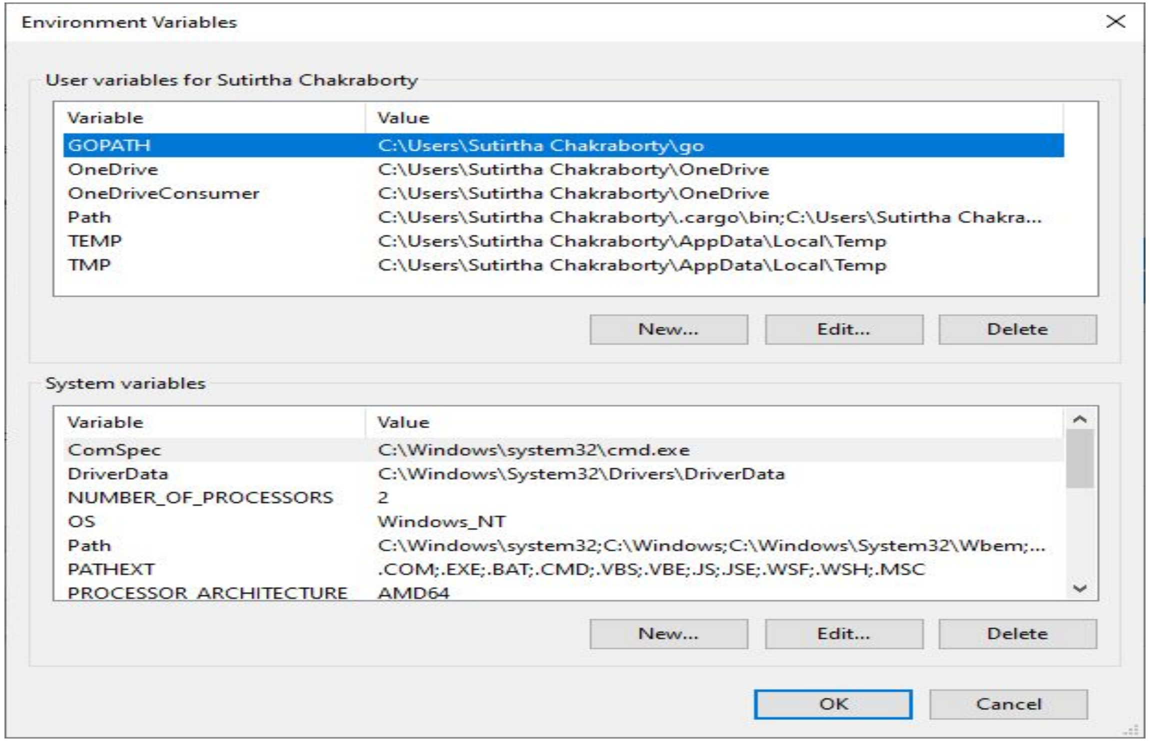
Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK Cancel Apply

Check if GOPATH is set up properly

➢ Click on **Environment Variables**



Environment Variables

User variables for Sutirtha Chakraborty

Variable	Value
GOPATH	C:\Users\Sutirtha Chakraborty\go
OneDrive	C:\Users\Sutirtha Chakraborty\OneDrive
OneDriveConsumer	C:\Users\Sutirtha Chakraborty\OneDrive
Path	C:\Users\Sutirtha Chakraborty\.cargo\bin;C:\Users\Sutirtha Chakra...
TEMP	C:\Users\Sutirtha Chakraborty\AppData\Local\Temp
TMP	C:\Users\Sutirtha Chakraborty\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
DriverData	C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS	2
OS	Windows_NT
Path	C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64

Check if GOPATH is set up properly

- ➢ Check if GOPATH is set
- ➢ Here in the image above **GOPATH** is set to **%USERPROFILE%\go**
- ➢ You can set it to any folder you prefer
- ➢ To change GOPATH just click **Edit** and then either write the path or browse the folder
- ➢ You have installed go in your Windows successfully

How to Install Go in Ubuntu 20.04

- ➢ We will be installing the latest version of **Go** which is **1.15.5**
- ➢ To download the latest version, go to the official download page and grab the tarball or use the following wget command to download it on the terminal

```
$ sudo wget https://golang.org/dl/go1.15.5.linux-amd64.tar.gz
```

- ➢ Next, extract the tarball to **/usr/local** directory

```
$ sudo tar -C /usr/local -xzf go1.15.5.linux-amd64.tar.gz
```

How to Install Go in Ubuntu 20.04

- ➢ Add the **go** binary path to **.bashrc** file **/etc/profile** (for a system-wide installation)

```
export PATH=$PATH:/usr/local/go/bin
```

- ➢ After adding the PATH environment variable, you need to apply changes immediately by running the following command

```
$ source ~/.bashrc
```

How to Install Go in Ubuntu 20.04

- ➢ Now verify the installation by simply running the **go** version in the terminal

```
$ go version
```

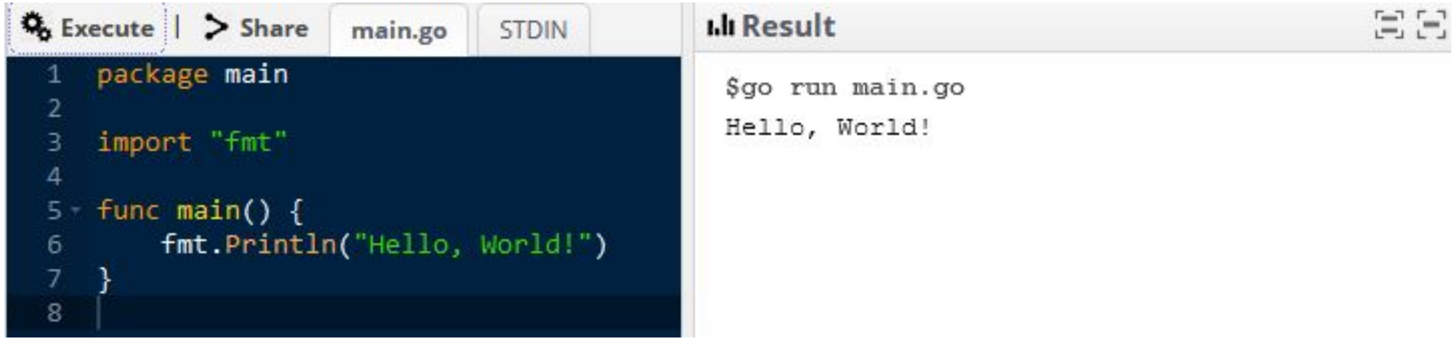
- ➢ You can also install **go** from the snap store too

```
$ sudo snap install --classic --channel=1.15/stable go
```

How to Install Go in Ubuntu 20.04

- ➢ Let's run **hello** world program
- ➢ Save the file with .go extension
- ➢ To run the program type **go** run <file-name> from the terminal

```
$ go run main.go
```



```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}
```

```
$go run main.go
Hello, World!
```

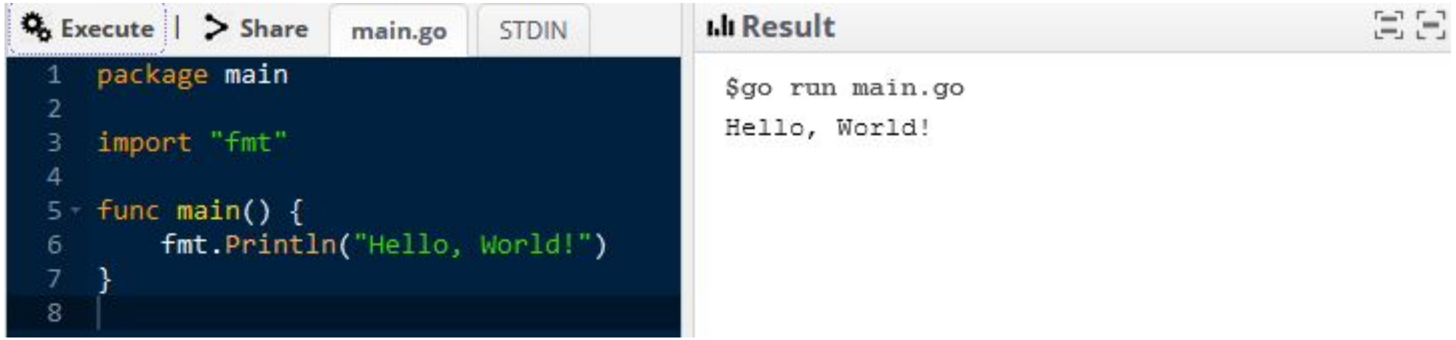
Create first application

- ➢ **Prerequisites**
- ➢ You will need a local Go development environment set up on your computer
- ➢ **Step 1 — Writing the Basic “Hello, World!” Program**
- ➢ To write the “Hello, World!” program, open up a command-line text editor such as nano and create a new file:

```
$ nano hello.go
```

Create first application

➢ Once the text file opens up in the terminal window, you'll type out your program:



```go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}

```

```
$go run main.go
Hello, World!
```

Create first application

- `package` is a Go *keyword* that defines which code bundle this file belongs to
- `import` is a Go *keyword* that tells the Go compiler which other packages you want to use in this file
- `fmt.Println` is a Go *function*, found in the fmt package, that tells the computer to print some text to the screen
- You follow the `fmt.Println` function by a sequence of characters, like `"Hello, World!"`, enclosed in quotation marks

Create first application

- ➢ Any characters that are inside of quotation marks are called a *string*
- ➢ The `fmt.Println` function will print this string to the screen when the program runs
- ➢ Save and exit `nano` by typing `CTRL + X`, when prompted to save the file, press `Y`

Create first application

- **Step 2 — Running a Go Program**
- Use the go command, followed by the name of the file you just created

```
$ go run hello.go
```

- The program will execute and display this output:

```
Hello, World!
```

Create first application

- **Step 3 — Prompting for User Input**
- Every time you run your program, it produces the same output
- In this step, you can add to your program to prompt the user for their name
- You’ll then use their name in the output
- Instead of modifying your existing program, create a new program called `greeting.go` with the `nano` editor:

```
$ nano greeting.go
```

Create first application

➢ First, add this code, which prompts the user to enter their name:

```
package main
import (
    "fmt"
)
func main() {
  fmt.Println("Please enter your name.")
}
```

➢ Once again, you use the `fmt.Println` function to print some text to the screen

Create first application

- ➢ Now add the highlighted line to store the user's input:

```
package main
import (
    "fmt"
)
func main() {
  fmt.Println("Please enter your name.")
  var name string
}
```

- ➢ The `var name string` line will create a new variable using the `var` *keyword*

Create first application

- You name the variable `name`, and it will be of type `string`
- Then, add the highlighted line to capture the user’s input:

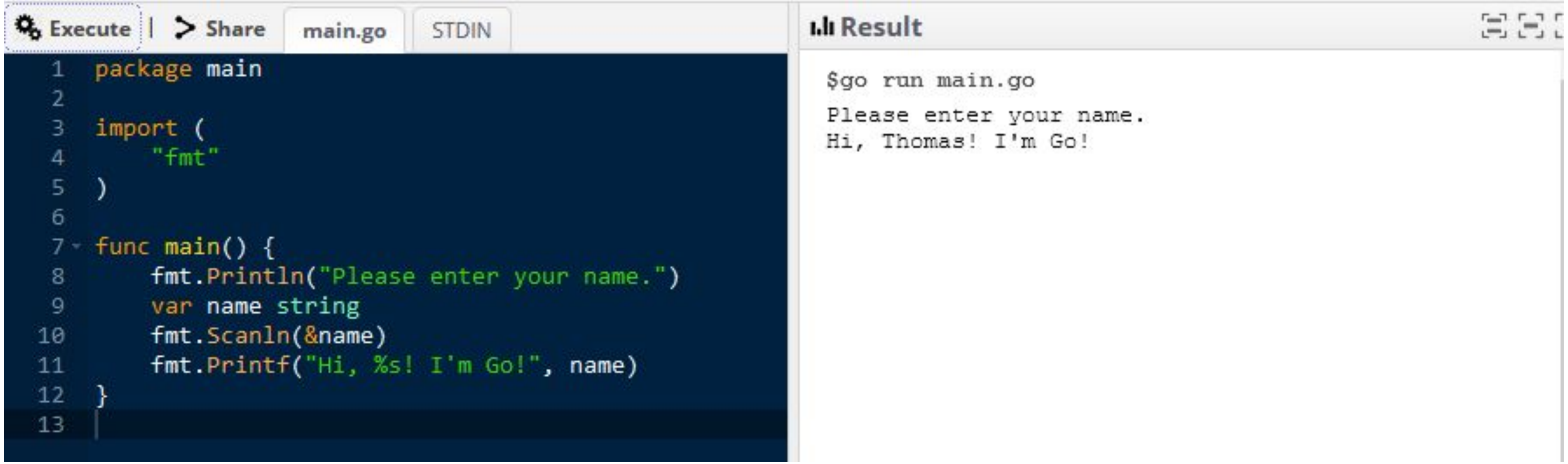
```
package main
import (
    "fmt"
)
func main() {
  fmt.Println("Please enter your name.")
  var name string
  fmt.Scanln(&name)
}
```

Create first application

- ➢ The `fmt.Scanln` method tells the computer to wait for input from the keyboard ending with a new line or (\n), character
- ➢ This pauses the program, allowing the user to enter any text they want
- ➢ The program will continue when the user presses the `ENTER` key on their keyboard
- ➢ All of the keystrokes, including the `ENTER` keystroke, are then captured and converted to a string of characters

Create first application

➢ Finally, add the following highlighted line in your program to print the output:



```go
package main

import (
	"fmt"
)

func main() {
	fmt.Println("Please enter your name.")
	var name string
	fmt.Scanln(&name)
	fmt.Printf("Hi, %s! I'm Go!", name)
}

```

```
$go run main.go
Please enter your name.
Hi, Thomas! I'm Go!
```

Create first application

- ➢ This time, instead of using the `fmt.Println` method again, you're using `fmt.Printf`
- ➢ The `fmt.Printf` function takes a string, and using special printing *verbs*, (%s), it injects the value of name into the string
- ➢ Save and exit `nano` by pressing `CTRL + X`, and press `Y` when prompted to save the file
- ➢ Now run the program
- ➢ You'll be prompted for your name, so enter it and press ENTER

Create first application

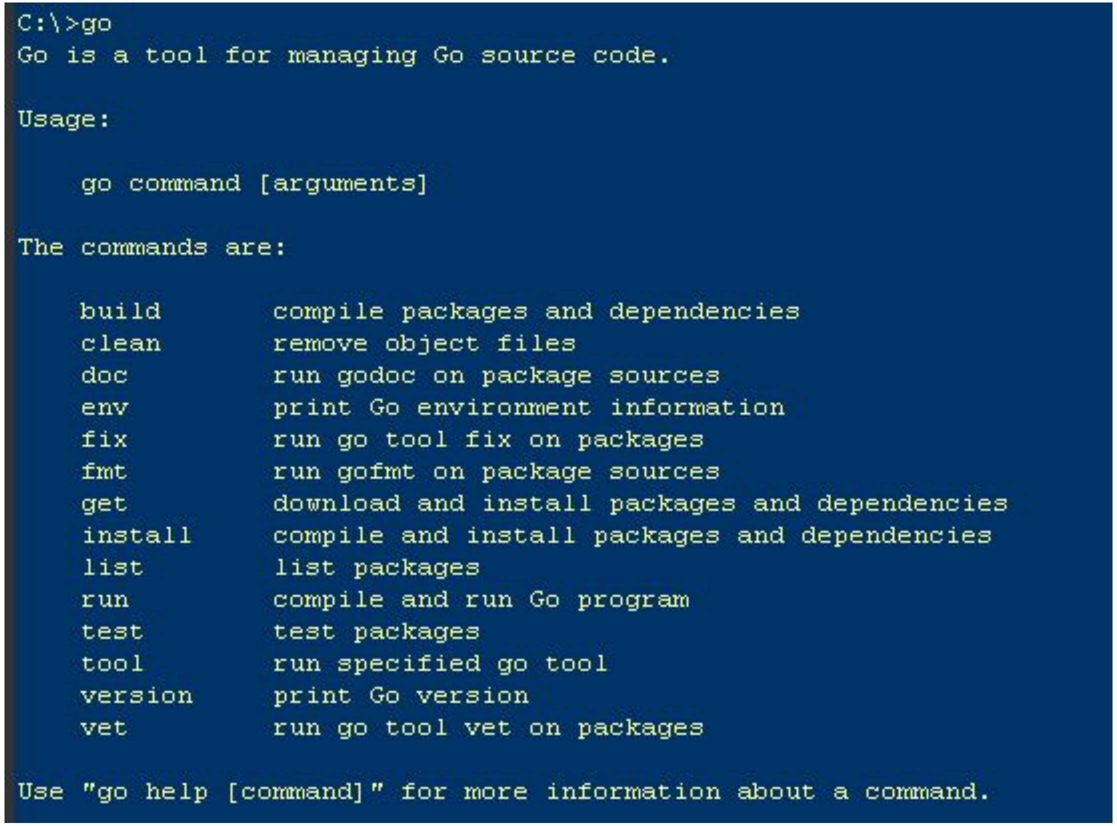
➢ Output

```
Please enter your name.
xyz
Hi, xyz
! I'm Go!
```

Common Go Commands

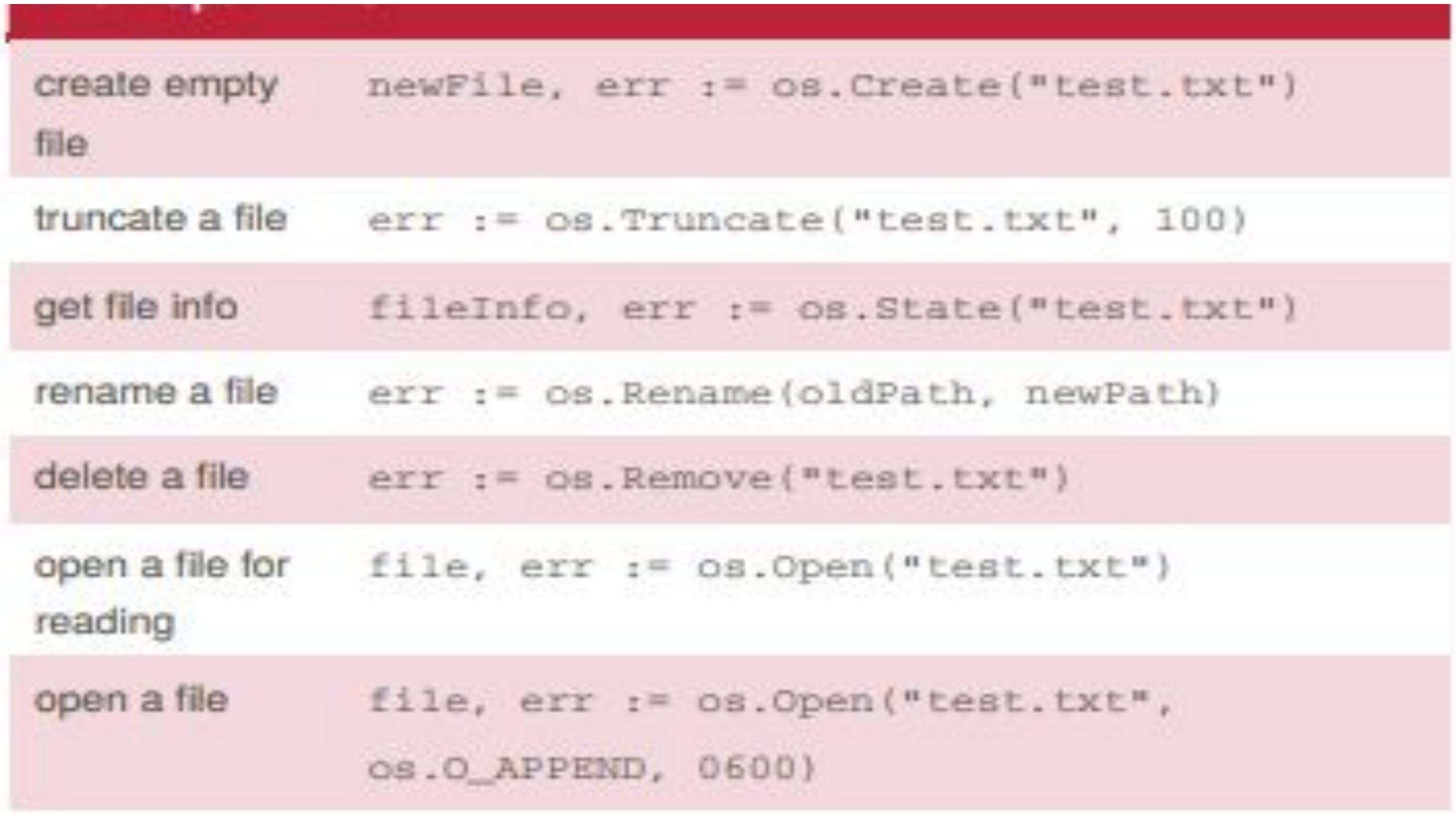
- Go Commands

Go Commands



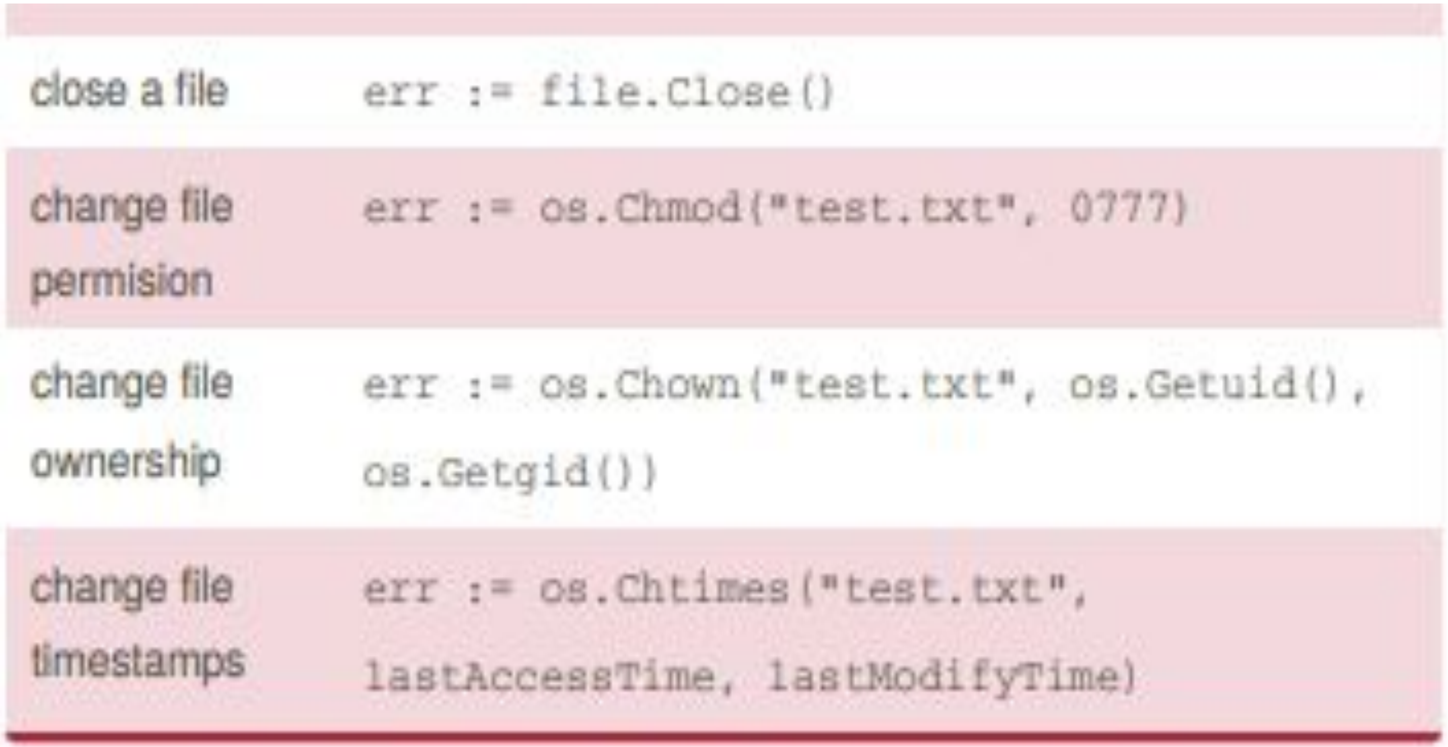
```
C:\>go
Go is a tool for managing Go source code.

Usage:

    go command [arguments]

The commands are:

    build       compile packages and dependencies
    clean       remove object files
    doc         run godoc on package sources
    env         print Go environment information
    fix         run go tool fix on packages
    fmt         run gofmt on package sources
    get         download and install packages and dependencies
    install     compile and install packages and dependencies
    list        list packages
    run         compile and run Go program
    test        test packages
    tool        run specified go tool
    version     print Go version
    vet         run go tool vet on packages

Use "go help [command]" for more information about a command.
```

Basic Operations



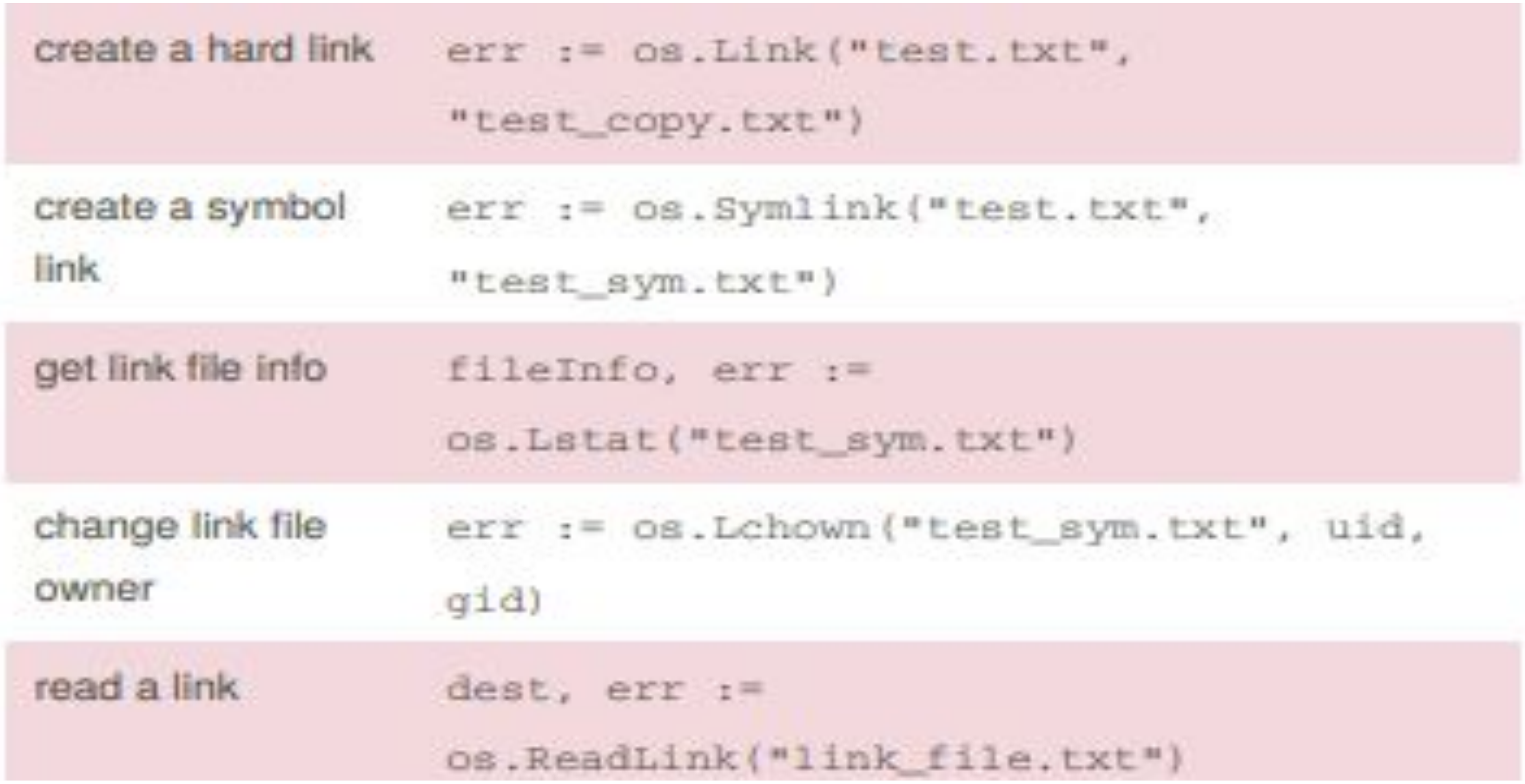
create empty file	`newFile, err := os.Create("test.txt")`
truncate a file	`err := os.Truncate("test.txt", 100)`
get file info	`fileInfo, err := os.State("test.txt")`
rename a file	`err := os.Rename(oldPath, newPath)`
delete a file	`err := os.Remove("test.txt")`
open a file for reading	`file, err := os.Open("test.txt")`
open a file	`file, err := os.Open("test.txt", os.O_APPEND, 0600)`

Basic Operations



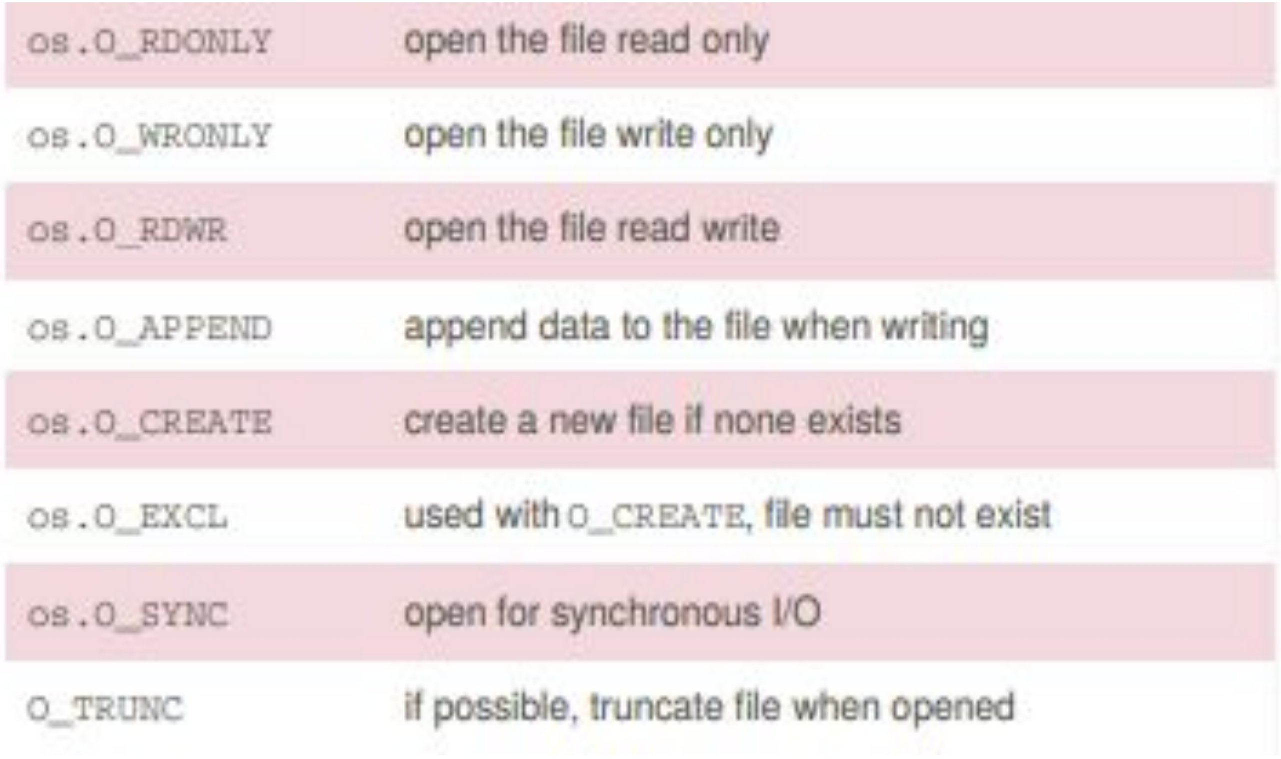
close a file	`err := file.Close()`
change file permision	`err := os.Chmod("test.txt", 0777)`
change file ownership	`err := os.Chown("test.txt", os.Getuid(), os.Getgid())`
change file timestamps	`err := os.Chtimes("test.txt", lastAccessTime, lastModifyTime)`

Hard Link & Symbol Link



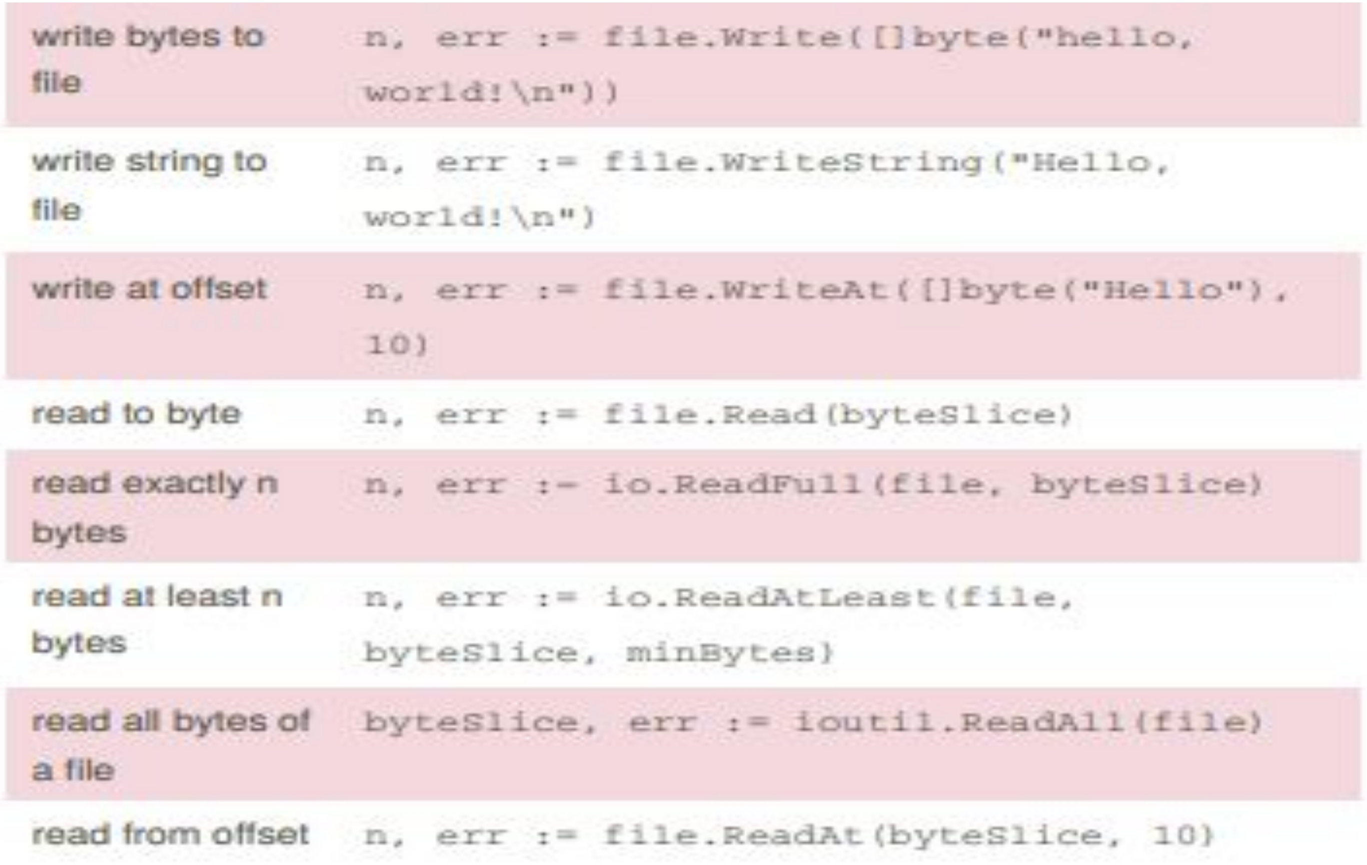
create a hard link	`err := os.Link("test.txt", "test_copy.txt")`
create a symbol link	`err := os.Symlink("test.txt", "test_sym.txt")`
get link file info	`fileInfo, err := os.Lstat("test_sym.txt")`
change link file owner	`err := os.Lchown("test_sym.txt", uid, gid)`
read a link	`dest, err := os.ReadLink("link_file.txt")`

file open flag



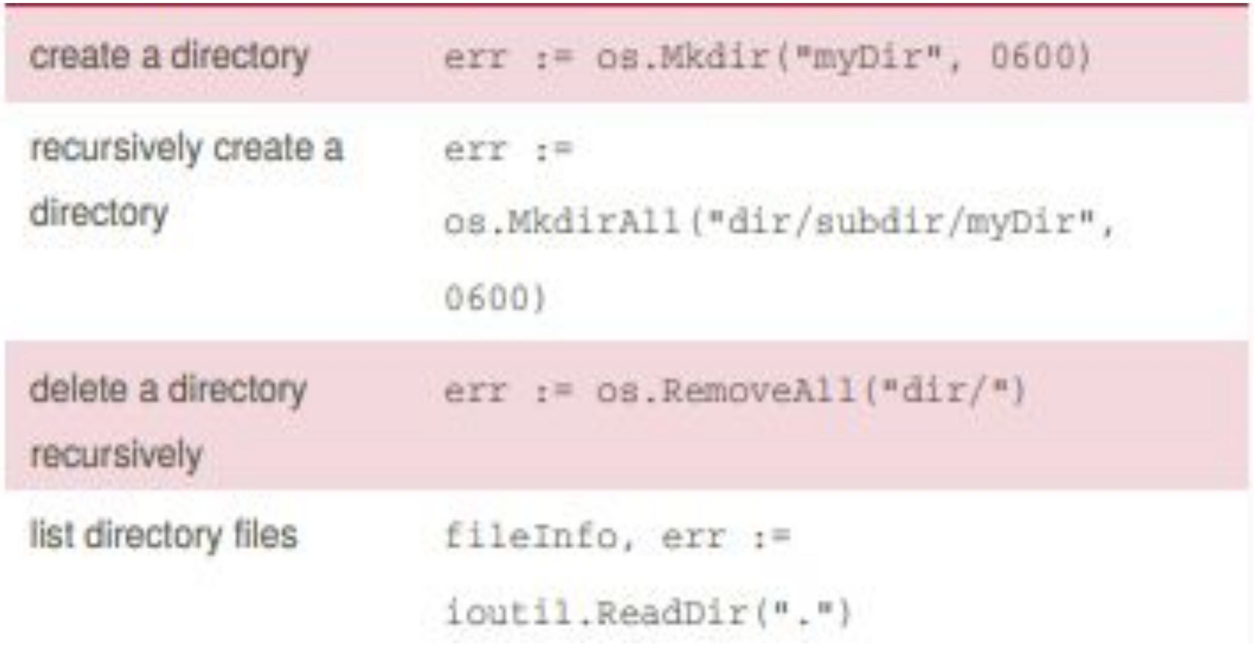
`os.O_RDONLY`	open the file read only
`os.O_WRONLY`	open the file write only
`os.O_RDWR`	open the file read write
`os.O_APPEND`	append data to the file when writing
`os.O_CREATE`	create a new file if none exists
`os.O_EXCL`	used with `O_CREATE`, file must not exist
`os.O_SYNC`	open for synchronous I/O
`O_TRUNC`	if possible, truncate file when opened

Read and Write



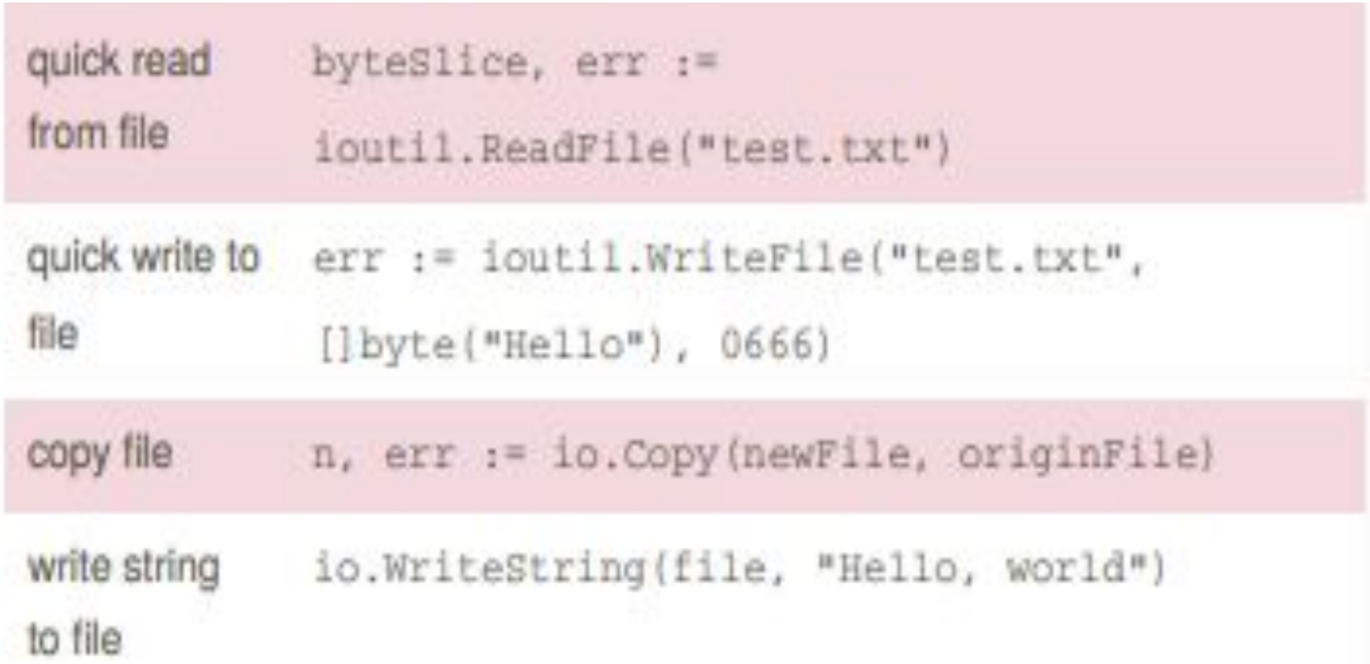
write bytes to file	`n, err := file.Write([]byte("hello, world!\n"))`
write string to file	`n, err := file.WriteString("Hello, world!\n")`
write at offset	`n, err := file.WriteAt([]byte("Hello"), 10)`
read to byte	`n, err := file.Read(byteSlice)`
read exactly n bytes	`n, err := io.ReadFull(file, byteSlice)`
read at least n bytes	`n, err := io.ReadAtLeast(file, byteSlice, minBytes)`
read all bytes of a file	`byteSlice, err := ioutil.ReadAll(file)`
read from offset	`n, err := file.ReadAt(byteSlice, 10)`

Work with directories



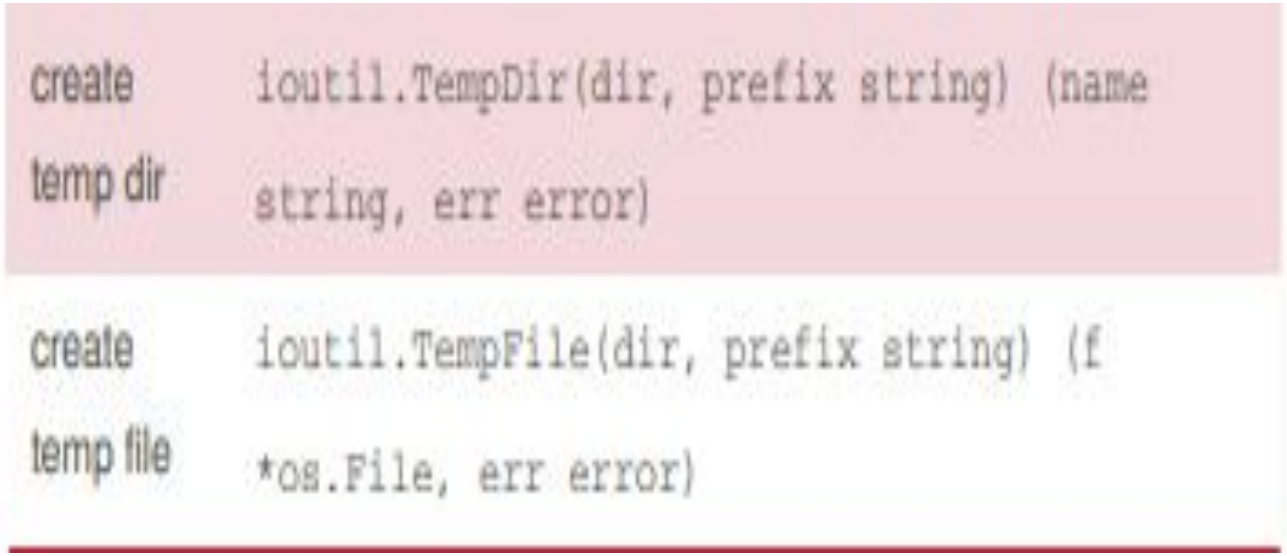
create a directory	`err := os.Mkdir("myDir", 0600)`
recursively create a directory	`err := os.MkdirAll("dir/subdir/myDir", 0600)`
delete a directory recursively	`err := os.RemoveAll("dir/")`
list directory files	`fileInfo, err := ioutil.ReadDir(".")`

Shortcuts



quick read from file	`byteSlice, err := ioutil.ReadFile("test.txt")`
quick write to file	`err := ioutil.WriteFile("test.txt", []byte("Hello"), 0666)`
copy file	`n, err := io.Copy(newFile, originFile)`
write string to file	`io.WriteString(file, "Hello, world")`

Shortcuts



create temp dir	`ioutil.TempDir(dir, prefix string) (name string, err error)`
create temp file	`ioutil.TempFile(dir, prefix string) (f *os.File, err error)`

Packages

- Packages in Go supports modularity, encapsulation, separate compilation, and reuse
- Package declaration at top of every source file
- Standalone executables program are in package main
- If an entity is declared within a function, it is local to that function
- If declared outside of a function, however, it is visible in all files of the package to which it belongs
- The case of the first letter of a name determines its visibility across package boundaries

Packages

- Uppercase identifier: Exported i.e visible and accessible outside of its own package
- Lower case identifier:
    - private (not accessible from other packages)

Pointers

- ➢ var x int = 11

  /*
- ➢ *int is intege rPo inter type.
- ➢ ‘p’ will contain the address of an integer variable.
- ➢ You can also say that p points to an int variable.

  */

  var p *int

Pointers

// Expression &var (address of var) yields a pointer to a variable.

p = &x // will contain address of x

// Expression *p points to the variable whose address p contains. *p is an alias for x.

fmt.Pr int ln(*p)

Arrays

- Arrays and Structs are aggregate type
- Arrays are homogeneous
- Array is fixed length sequence of zero or more elements of particular type
- var a[3] int // Array of 3 integers
- var a[3]int = [3]int{1, 2, 3} // use an array literal to initialize an array with a list of values
- a[len(a)-1] // Print last element
- q := [...]i nt{1, 2, 3} // with ellipsis ... , array length is determined by the

Declarations

- ➢ There are four major kinds of declarations: var, const, type, func
- ➢ **var**
- ➢ var name type = expression

  // Either the type or the =expre ssion part may be omitted, but not both

  // If the type is omitted, it is determined by the initia lizer expres sion. If the expression is omitted, the initial value is the zero value for the type, which is 0 for numbers, false for booleans.

Declarations

var foo int = 42 // declare and init. var name type = expression

var sep string // implicit initialize

- s, sep := " ", " " // Short variable declar ation. name := expression
- p := new(int) // p, of type *int, points to an unnamed int variable

  // new(T) creates unnamed variable of type T, initialize it to the zero value of T and returns its address.

Declarations

- **const**
- A constant is an identifier for a fixed value
- The value of a variable can vary, but the value of a constant must remain constant

  const constant = "This is a consta nt"

  const a float64 = 3.14

Declarations

- ➢ **Function Declaration**
- ➢ A function declaration has a name, a list of parame ters, an optional list of results
- ➢ // function with params

  func getFul lNa me(fir stName string, lastName string) {}
- ➢ // Multiple params of the same type

  func getFul lNa me(fir stName, lastName string) {}
- ➢ // Can return type declar ation

  func getId() int

Declarations

- // Can return multiple values at once

```
func person() (int, string) {
return 23, " vin ay"
}
```

- // Can return multiple named results

```
func person() (age int, name string) {
age = 23 name = " vin ay"
return
}
```

Declarations

- var age, name = person()

  // Can return function

  func person() func() (strin g,s tring) {

  area:= func() (strin g,s tring) {

  return " str eet ", " cit y"

  }

  return area

  }

Features Of Golang

- ➢ Language is very concise, simple and safe
- ➢ Compilation time is very fast
- ➢ Patterns which adapt to the surrounding environment similar to dynamic
- ➢ languages
- ➢ Inbuilt concurrency such as lightweight processes channels and select statements
- ➢ Supports the interfaces and the embedded types

Lack of essential features

- No ternary operator ?:
- No generic types
- No exceptions
- No assertions
- No overloading of methods and operators

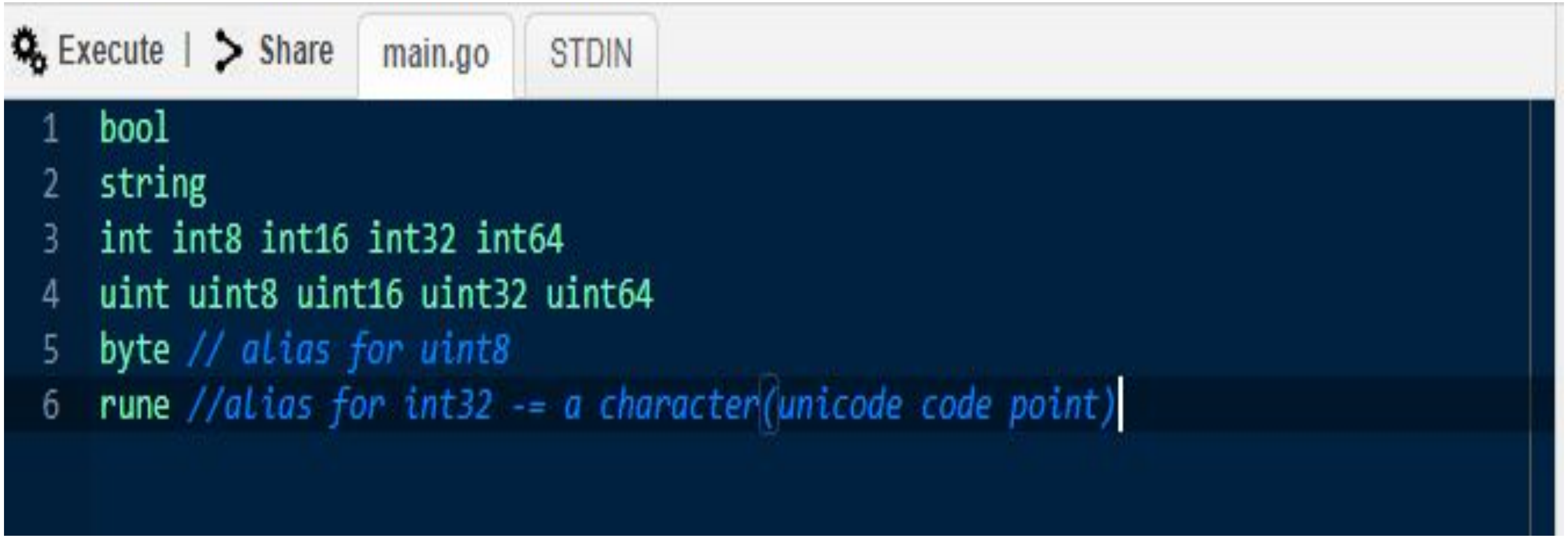
Companies Using Golang

- ➢ Google for “dozens of systems”
- ➢ Docker a set of tools for deploying linux containers
- ➢ Openshift a cloud computing platform as a service by Red Hat
- ➢ Dropbox migrated few of their critical components from Python to Go
- ➢ Netflix for two portions of their server architecture
- ➢ Soundcloud for “dozens of systems”
- ➢ ThoughtWorks some tools and applications around continuous delivery and instant messages (CoyIM)

Companies Using Golang

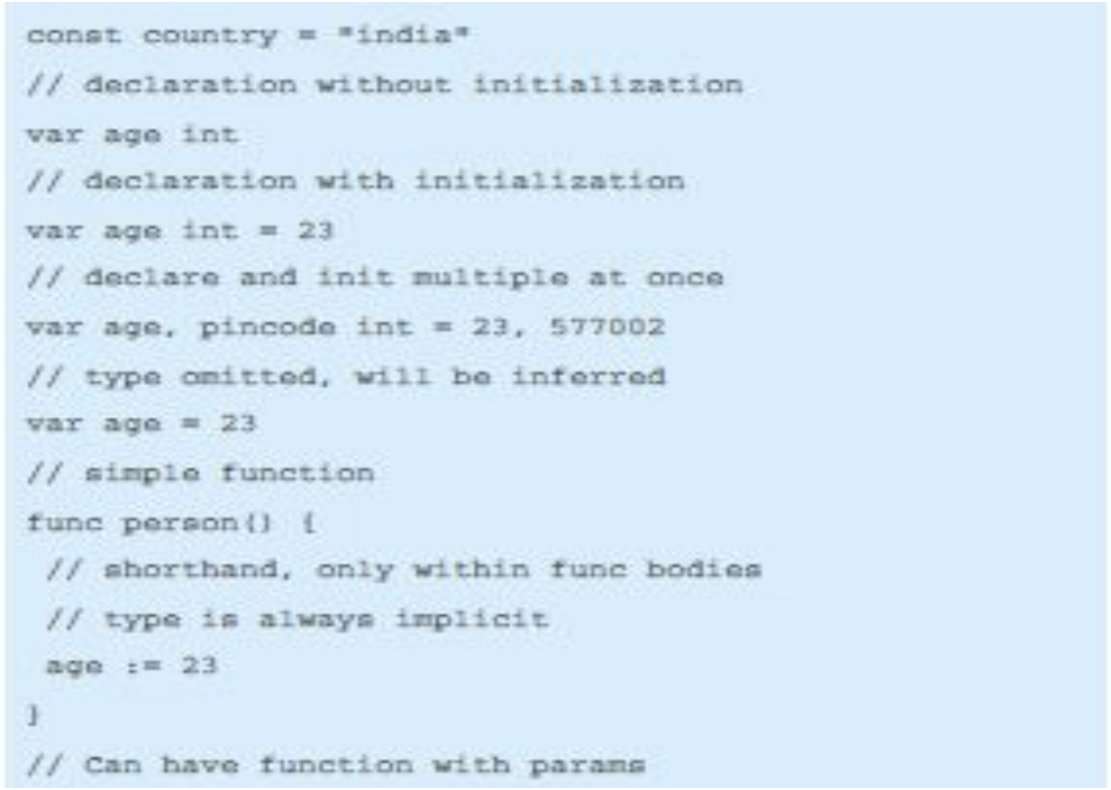
- ➢ Uber for handling high volumes of geofence- based queries.
- ➢ BookMyShow for handling high volume of traffic, rapidly growing customer, to adapt new business solution and (cloud solution) distribution tools

Built-in Types



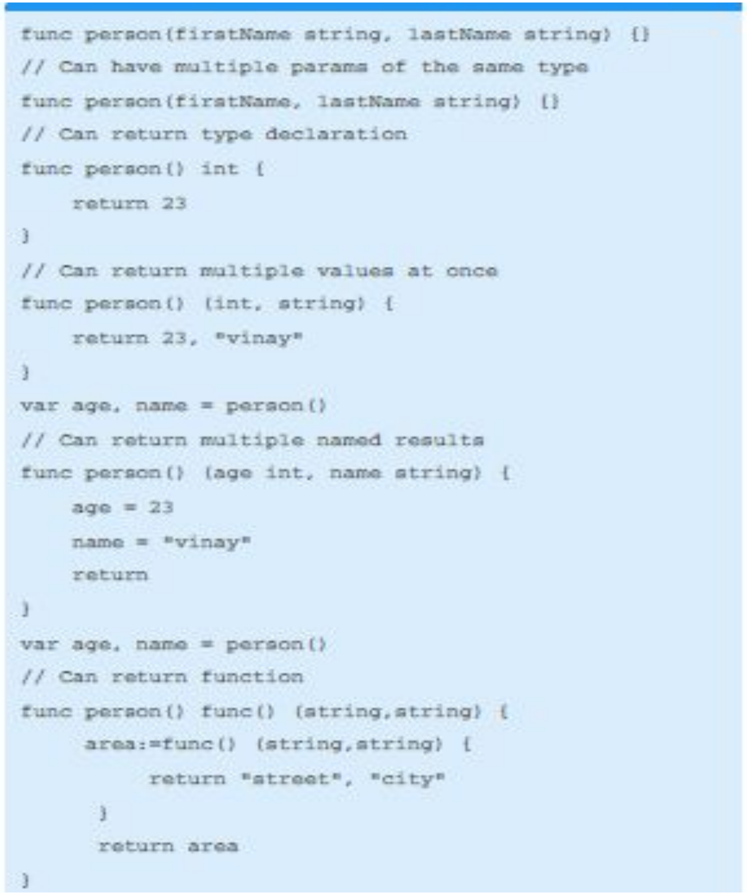
```
bool
string
int int8 int16 int32 int64
uint uint8 uint16 uint32 uint64
byte // alias for uint8
rune //alias for int32 -= a character(unicode code point)
```

Variable & Function Declarations



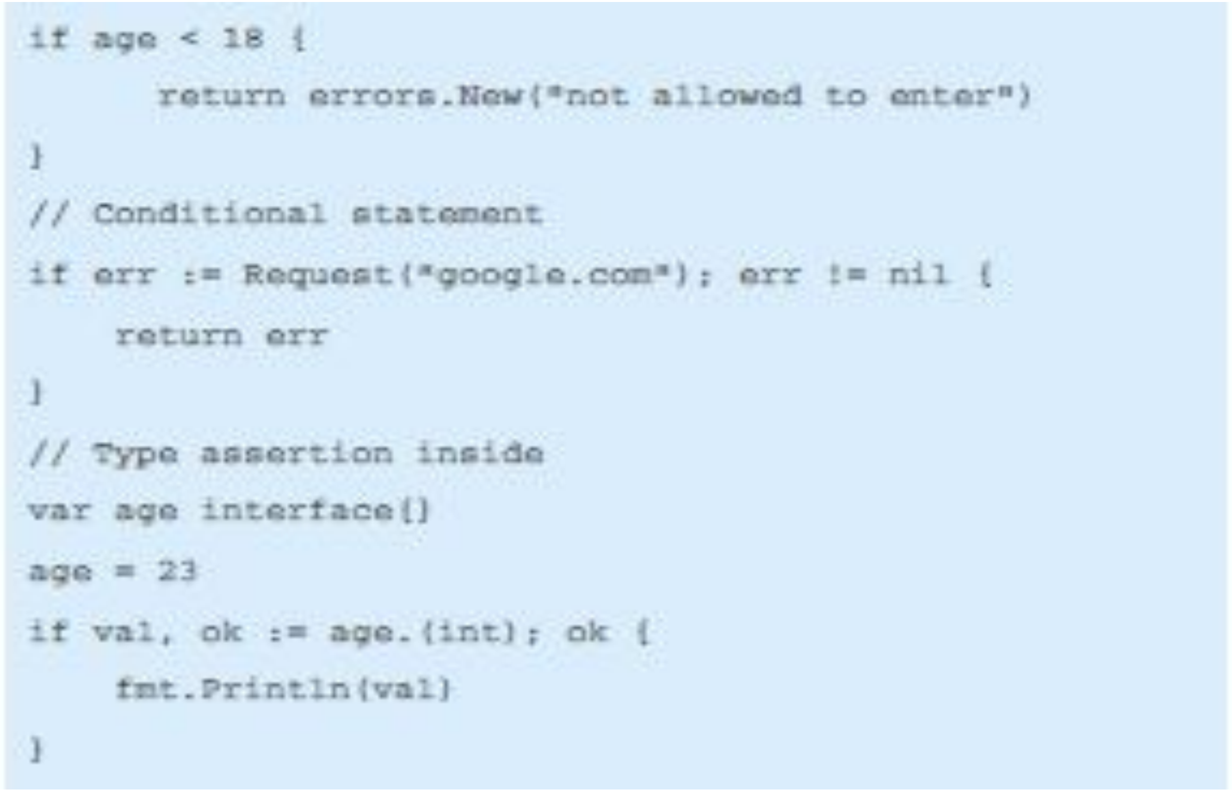
```
const country = "india"
// declaration without initialization
var age int
// declaration with initialization
var age int = 23
// declare and init multiple at once
var age, pincode int = 23, 577002
// type omitted, will be inferred
var age = 23
// simple function
func person() {
 // shorthand, only within func bodies
 // type is always implicit
 age := 23
}
// Can have function with params
```

Variable & Function Declarations



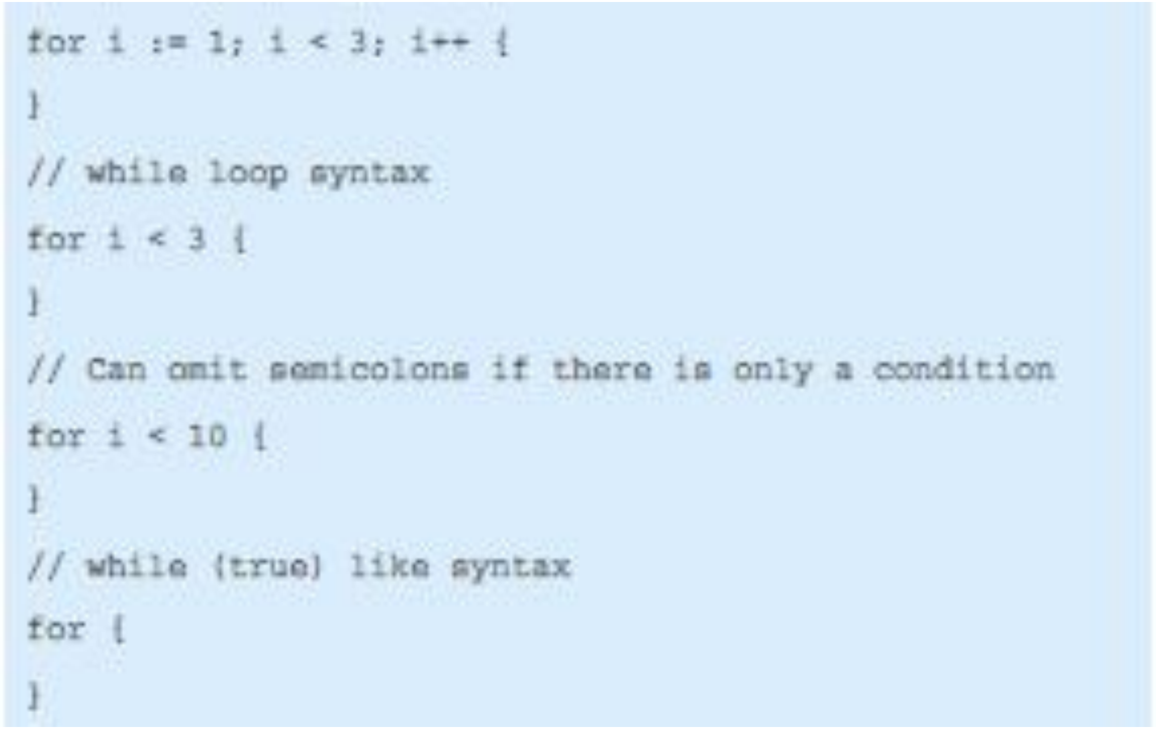
```
func person(firstName string, lastName string) {}
// Can have multiple params of the same type
func person(firstName, lastName string) {}
// Can return type declaration
func person() int {
    return 23
}
// Can return multiple values at once
func person() (int, string) {
    return 23, "vinay"
}
var age, name = person()
// Can return multiple named results
func person() (age int, name string) {
    age = 23
    name = "vinay"
    return
}
var age, name = person()
// Can return function
func person() func() (string,string) {
     area:=func() (string,string) {
          return "street", "city"
     }
     return area
}
```

If statement



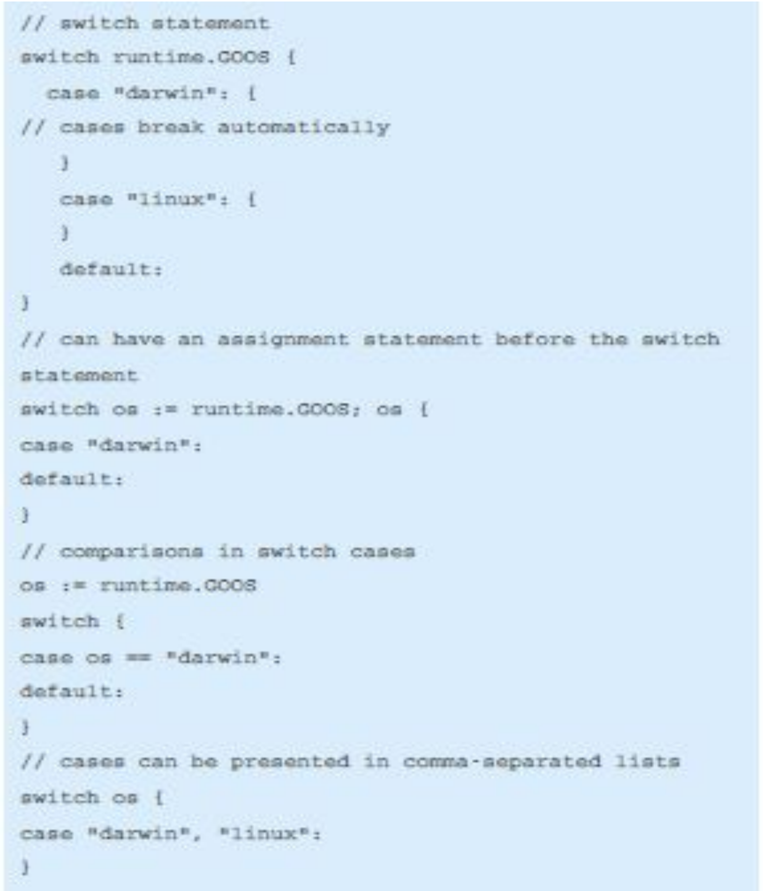
```
if age < 18 {
        return errors.New("not allowed to enter")
}
// Conditional statement
if err := Request("google.com"); err != nil {
    return err
}
// Type assertion inside
var age interface{}
age = 23
if val, ok := age.(int); ok {
    fmt.Println(val)
}
```

Loop statement



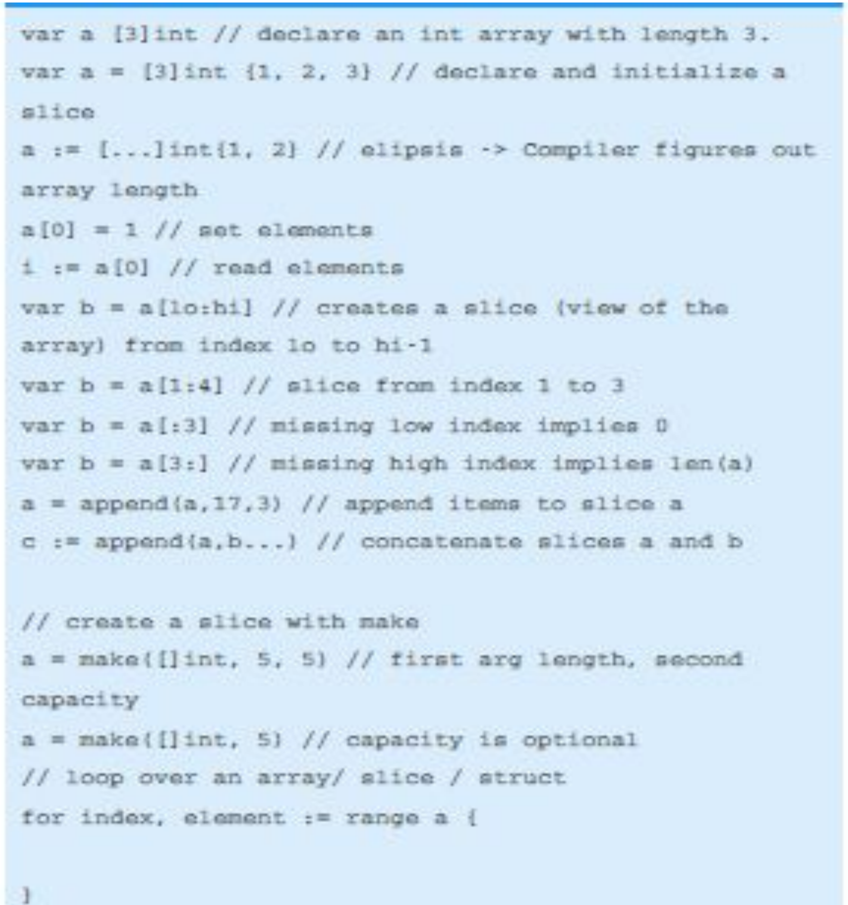
```
for i := 1; i < 3; i++ {
}
// while loop syntax
for i < 3 {
}
// Can omit semicolons if there is only a condition
for i < 10 {
}
// while (true) like syntax
for {
}
```

Switch statement



```
// switch statement
switch runtime.GOOS {
  case "darwin": {
// cases break automatically
   }
   case "linux": {
   }
   default:
}
// can have an assignment statement before the switch
statement
switch os := runtime.GOOS; os {
case "darwin":
default:
}
// comparisons in switch cases
os := runtime.GOOS
switch {
case os == "darwin":
default:
}
// cases can be presented in comma-separated lists
switch os {
case "darwin", "linux":
}
```

Arrays, Slices



```
var a [3]int // declare an int array with length 3.
var a = [3]int {1, 2, 3} // declare and initialize a
slice
a := [...]int{1, 2} // elipsis -> Compiler figures out
array length
a[0] = 1 // set elements
i := a[0] // read elements
var b = a[lo:hi] // creates a slice (view of the
array) from index lo to hi-1
var b = a[1:4] // slice from index 1 to 3
var b = a[:3] // missing low index implies 0
var b = a[3:] // missing high index implies len(a)
a = append(a,17,3) // append items to slice a
c := append(a,b...) // concatenate slices a and b

// create a slice with make
a = make([]int, 5, 5) // first arg length, second
capacity
a = make([]int, 5) // capacity is optional
// loop over an array/ slice / struct
for index, element := range a {

}
```