Итак, давайте посмотрим на самый простой пример запуска приложения. И сначала разберемся с самим, значит, exec. Итак. Давайте посмотрим. Значит, вот у нас здесь есть package exec. Он в себя много чего импортирует. Нас это не особо интересует. Имеет какие-то ошибки. И самое главное – это команда, структура Cmd, которая представляет собой внешнюю команду для выполнения.

Значит, Cmd в себя включает путь к приложению, аргументы приложения, какие-то переменные окружения приложения, мы еще к ним вернемся. Директорию приложения, включает в себя вот эти все reader-ы и writer-ы для…, значит, stdin, stdout приложения. Включает в себя файлы, процессы, которые запускаются приложением, ну и так далее. И соответственно, первое, что нас интересует, это функция command, которая является конструктором, собственно говоря, возвращающим в себя ссылку на структуру Cmd. Базовая команда включает в себя имя и список аргументов приложения. И из этого просто возвращает Cmd.

Давайте посмотрим, как это выглядит. Соответственно, вот у меня есть package exec, который я сюда проимпортировал os/exec. И в нем есть command, в command я просто включаю name. То есть, если бы мне надо было запустить какое-то локальное приложения, я бы включал сюда полный путь, значит, к своему приложению. И соответственно, команда run, которая…, ну соответственно, метод, структура Cmd, которая просто запустит, значит, эту Cmd. Под капотом у нас просто метод Start, который берет descriptor-ы, проверяет, если мы не window запущены. И собственно говоря, делает всю эту магию, на которой я подробно останавливаться не буду, по сути, это Wrapper над другим системным вызовом, os.StartProcess, который, в свою очередь, уже обращается к системным атрибутам. К атрибутам системы для того, чтобы уже сделать системный вызов syscall, так называемый, для вызова процесса. Мы так глубоко погружаться не будем, нам нужно просто знать, соответственно, интерфейс для запуска приложения.

Ну и давайте посмотрим, как это работает. Соответственно, я вызываю браузер firefox, с помощью команды Cmd.run. Обратите внимание, что сейчас приложение зависло, но, то есть как зависло, оно ждет сейчас запуска descriptor-а процесса, потому что в Cmd run встроено у нас… Да, вот смотрите, Firefox запустился. Firefox так и будет висеть. Здесь в Cmd run у нас встроен метод wait, который ждет запуска приложения, ждет передачи, ждет передачи descriptor-а приложения. То есть пока я не закрою firefox, либо не закончу приложение, они будут вместе работать. Я могу закрыть firefox сейчас, и мое приложение завершилось, соответственно, без ошибок, потому что err у нас пустой, то есть firefox нормально вышел.

Давайте теперь посмотрим несколько более сложные примеры. Давайте посмотрим на базе сложный пример. Хотелось бы, кстати, в тему моего предыдущее видео обратить ваше внимание на то, что функции внутри package-и, которые вы планируете использовать снаружи, пишутся с большой буквы. Функции, которые пишутся с маленькой буквы, считаются в go приватные. Потому что как такового разделения – приватных и публичных функций в go нет. Поэтому он скрывает просто функции с маленькой буквы. И позволяет вам использовать функции с большой буквы. Просто во всех этих, значит, примерах, я называю функции каким-то образом, а потом из своего main go я, из соответствующего package, их вызываю. Так, в принципе, будут устроены почти все практические примеры golang-а.

Итак, мы хотим запустить приложение с аргументами. В данном случае мы взяли приложение tr, которое транслирует строки, может из них что-то вырезать, может что-то добавить, может, в нашем случае, преобразовать строку. Мы хотим преобразовать строку из маленьких букв латинского алфавита «A-Z», в большие буквы латинского алфавита. И что нам для этого нужно? Нам нужно передать параметры. То есть помимо аргументов функции, мы должны туда передать

какой-то ввод. И это делается с помощью, значит, reader-а, который расположен в stdin. Я пользуюсь стандартным reader-ом строк, и просто передаю туда строку. То есть можно видеть, что интерфейс Cmd устроен в духе как бы вы работали с командой в консоли, с точки зрения самой команды. То есть команда читает stdin, и что-то отдает в stdout. В stdout, соответственно, в структуре команды определен writer. Writer – это какой-то поток байтов, который, значит, вы должны как-то записать и хранить. И у writer-а есть много имплементаций. То есть самое главное для writer-а, это, по сути, говоря, метод write. Это просто один интерфейс одним, по сути, методом.

И для наших целей очень хорошо подойдет, например, bytes.buffer, который имеет в себе метод write. Вот он. И соответственно, этот buffer, чем он нам ценен, байтовый, тем, что он может сразу отдать строчку, и мы будем как бы знать, что нам программа отдала. Ну и соответственно, здесь уже знакомый Cmd.run. То есть он как раз будет читать stdin, записывать в stdout. Проверяем, что у нас ничего не свалилось. Если ничего не свалилось, у нас значит…, мы вводим фразу, которая у нас получилась. Давайте посмотрим. Тут уже, в принципе, видно, пример выполнения, то, что запускал до этого. И да, вот, little slurm goes big, маленький slurm-ик, значит, стал большим. Вот так работает передача, значит, аргументов.

Давайте посмотрим на другие примеры. Можно увидеть, что, если мы просто запустим команду tr, то есть она будет ждать ввода нашего какого-то пользовательского. И вот здесь только будет выводить. Но есть, естественно, в Linux-е куча консольных приложений, которые не ждут никакого-то поискового ввода, которым просто можно передавать переменные. Переменные можно передавать следующим образом, то есть несколько переменных может передавать программа следующим образом, просто через запятую. Это, так называемый, var args, который существует, я уж даже не знаю, почти везде. То есть это вот эти три точки, они обозначают, что передать переменные вы можете

сколько хотите раз, и они внутри уже объединяются в, соответственно, массив. Немного функциональщины от go.

Соответственно, мы здесь просто возьмем команду echo. И выведем три строчки в них, просто передав их. Тут уже, в принципе, у меня все готово. Давайте посмотрим. Да. There are slurms in slurmland. В общем, все вот так вот работает. Можно заметить, что здесь добавилась новая команда output. Если помните, мы в предыдущем шаге писали, значит, все в буфер, в stdout. Выводили оттуда String-и. В принципе, go это делает за нас. И соответственно, уже тут есть некая обработка ошибок. И существует уже возврат готовой ошибки, go-шной. И существует возврат просто состояния, просто как бы того, что выдала функция. То есть мы здесь просто его можем скастить в строку, в массив байтов. И напечатать.

То же самое можно делать для функции, для любой функции. То есть можно это даже (НРЗ 08:18) вызывать. Как видите, я сразу здесь вывожу out-ы и ошибки. Давайте посмотрим, как это работает с нашей всеми известной командой ls -l. И вот мы видим, что здесь вот просто выведен…, прям как она это, значит…, как эта программка, как эта утилитка выводит, так оно все и выводиться из текущей папке. Соответственно, достаточно удобно, если вам просто нужно что-то передать или запечатлеть вызов.

Иногда существуют моменты, когда вам нужно не просто запустить какое-то приложение, а запустить и так, чтобы оно висело отдельным процессом и ждало, например, ввода какого-то вашего другого процесса внутри вашей системы. Чтобы реализовать такой функционал, мы используем так называемый, piping. То есть мы открываем pipe в stdin. Это, так называемый, input type. Ну то есть кто с Linux-ом работал, тот знает, что это такое. И соответственно, эта вещь позволяет нам просто набрасывать внутрь приложения различные команды.

То есть здесь вот, что у нас происходит? Мы значит открыли stdin pipe. Он будет открыт пока приложение не выйдет, пока go не поймет, что приложение вышло. Как и любой pipe, он должен быть закрыт. И соответственно, мы здесь, у нас есть команда, мы к ней открыли pipe, она запустилась. И мы здесь делаем, значит, goroutine-у, используя, значит, writestring из обычного пакета – io. То есть это обычный pipe, в котором мы можем писать обычные данные. Мы передаем туда строчку в команду cat, который потом ее и выведет. И затем выводим, так называемый, combinedoutput. То есть совмещаем ошибки вместе с выводом этой программы.

Вот. Давайте посмотрим, как это работает. Собственно говоря, здесь видно, как это работает. Нет ошибок. И соответственно, наша строчка.

Итак, если в предыдущем примере с входным pipe-ом, мы запускали приложение здесь. То в случае с выходным pipe-ом, когда нам нужно прочитать, мы будем использовать другой метод. Мы будем использовать smd.Start. Потому что здесь мы не делаем combinedoutput, нам нужно запустить приложение, прочитать все, что пишется в выходной pipe, подождать пока приложение, соответственно, завершиться. И после этого вывести, что у нас там написано в pipe-е.

То есть удобно, если у вас, например, есть какой-то logger. Вам нужно запустить в какой-то goroutine-е или в главном потоке, и ждать, пока он там что-то выполняется. И что-то там делает.

Соответственно, для этого вы, конечно, не можете сразу взять, сделать combinedoutput, завершить приложение и все. То есть вам нужен какой-то поток, который вы сможете читать. В данной случае у меня echo, поэтому поток сразу кончиться. Запускаем, соответственно, наш поток. И вот видим piping slurms. Ну то

есть все echo прочитало. Cmd.waite закончился. И соответственно, в pipe все у нас записалось.

В принципе, это, наверное, реально все, что можно делать с приложениями. Можно видеть, что, как входы, так и выходы, у нас контролируются, нам доступны. Соответственно, запекаются все с помощью команды Cmd. Все есть внутри, соответственно, package-e exec. И в принципе, тут, наверное, мне добавить больше нечего.