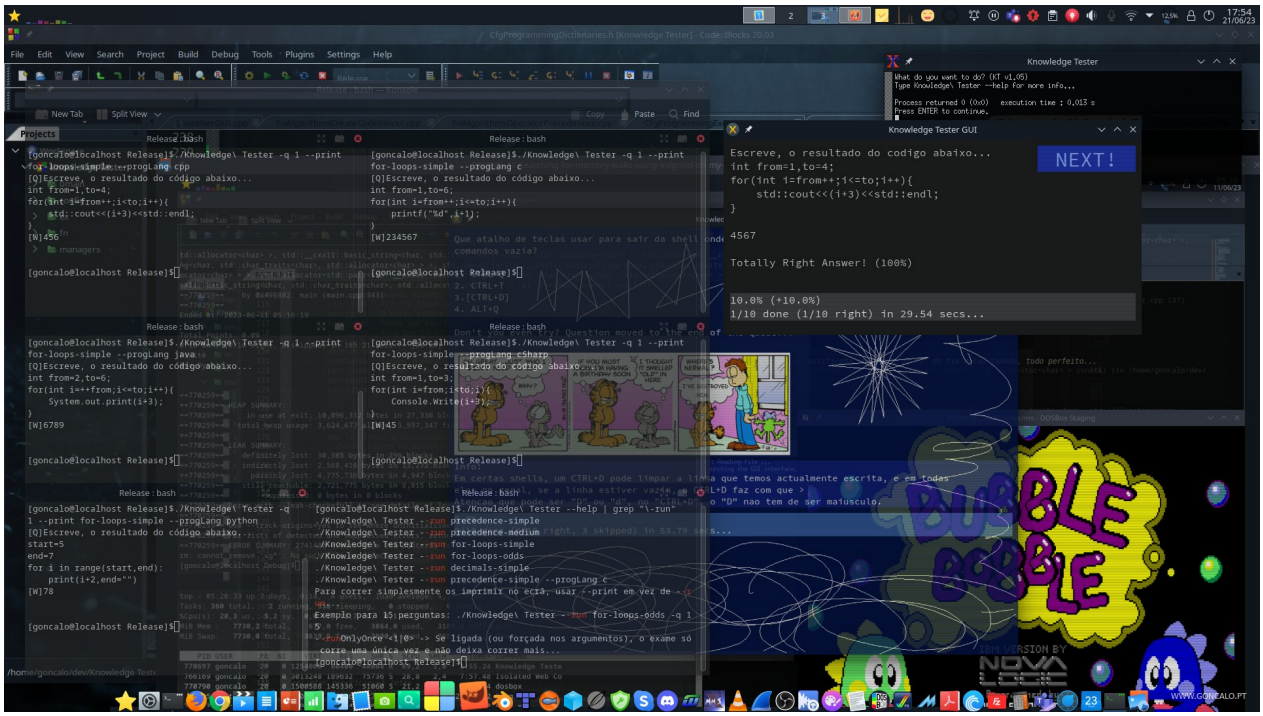
Tutorial *Knowledge Tester v1.10*



Para *Linux*

(Terminal & *GUI*)

2023-09-03

Índice

Table of Contents

Sobre este manual

Este manual não é muito pequeno, mas deve-se à quantidade enorme de opções e funcionalidades deste *software*, e escrevi-o com alguma pressa, pelo que poderão não ter sido mencionadas algumas funcionalidades devido a esquecimento, e ter uma escrita apressada e pouco elaborada, e até poderão faltar neste manual serem referidas algumas funcionalidades.

Foi também escrito à pressa em Agosto de 2023, e nunca lido/corrigido, por isso ignorem erros de escrita que possam aparecer.

Sobre o Knowledge Tester

O ***KT*** é um *software* que tanto pode ser usado para formadores ou formandos, ou mesmo qualquer outra pessoa que queira aprender em casa ou praticar, pode ser usado para auto-estudo, bem como para ensino/formação, seja presencial e remoto, e até para execução de exames remotos protegidos por encriptação *Rijndael 256 bits (AES 256)*, que desenvolvi nas minha linguagem de programação favorita (*C++*), para o meu sistema operativo favorito (*Linux)*.

Este *software* permite a que exista uma aprendizagem muito acelerada através da repetição e memorização.

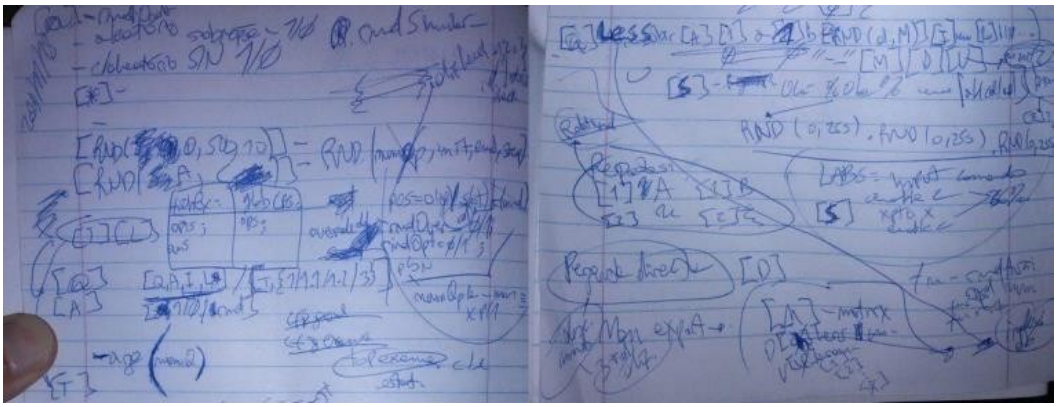
Em 2023 decidi migrá-lo para *Windows*, e a forma mais fácil, dado que o terminal do *Windows* não segue muito as normas, foi criar uma interface gráfica (*GUI*) para o meu *software*, e que já está a ver ser desenvolvido em ambiente gráfico, e em breve será migrado para *Windows* também.

Como decidi criar a sua *GUI* do zero, píxel a píxel, sem usar librarias famosas como a *Qt*, deixei o programa funcional em 2023 no modo gráfico mas não tive tempo para migrar o *software* para *Windows*, e vou deixar isso para 2025, por ter outros projectos importantes em mente entretanto e não querer dedicar todo o meu tempo livre de programação a este projecto sozinho.

A versão de consola será apenas para *Linux*, que terá tanto versão gráfica como terminal, e o *Windows* terá apenas a versão gráfica, com design 100% igual à do *Linux*, dado que eu faço tudo do zero.

A ideia deste programa nasceu de eu querer criar algo que permitisse a qualquer pessoa assimilar muita matéria de forma rápida, e comecei pelo modo de terminal porque tive há poucos anos, em 2019, uma colega invisual, que tinha dificuldades em aprender certas matérias, mas que tinha mais facilidade em usar o *Linux*, devido às muitas ferramentas que tinha, em especial *text-to-speech* em *shells*, e para a ajudar também, comecei a desenvolver o *software*, que teve como origem a folha abaixo de rascunhos escrita numa dessas aulas.

Como se pode ver pela imagem abaixo, o sistema foi imaginado de forma muito simples, baseado em *tags*, conforme explicado abaixo na secção sobre criação de exames:



Podem reparar nas tags “[Q]” (de “*Question*”), “[I]” (de “*Info*”), as “[1]” e “[0]” para perguntas certas e erradas, etc, um pequeno esboço que criei durante uma hora de formação que tive em 2019.

É facílimo criar rapidamente em modo de texto, perguntas e respostas para este *software* para depois poderem praticar as mesmas.

Neste tutorial explicarei por alto como usar o *software* para auto-estudo, para criação de exames, para treinar para certificações, como encriptar exames, criar exercícios, etc.

Este é um manual inicial (a segunda versão do mesmo, o original de 2021-11-15 só tinha 64 páginas), com o tempo à medida que o programa evolua, ele será também melhorado.

O sistema de *Tags*

Este *software* é muito baseado num sistema de *tags*, existindo tags como [Q], [S], [W], [1], [0], [I], [L], e mais tarde serão adicionadas outras, mas de forma simples, a proporcionar que qualquer pessoa possa criar exames em puro texto, num simples bloco de notas.

Mais tarde serão adicionadas outras à medida que o *software* vá sendo desenvolvido.

O uso do *Knowledge Tester* para auto-estudo

Este *software* pode ser usado para auto-estudo em vários níveis, e é especialmente útil para quem estuda para certificações de *IT*, que como todos sabemos, se baseiam muito em “testes de cruzinhas”, mas permite também respostas directas multi-linham, e até pequenos laboratórios com um seguimento lógico de respostas na mesma pergunta, para quem queira praticar para certificações práticas como por exemplo a *Red Hat Certified Engineer*, entre outras.

Ele pode ser usado também noutras áreas e disciplinas, pois permite tanto com perguntas directas como cruzes, entre outras funcionalidades a desenvolver no Futuro, bem como criarmos os nossos próprios exames.

E mesmo nas de resposta directa, permite múltiplas opções certas por linha, como “grep fich.txt pato” e “cat fich.txt | grep pato” em simultâneo.

O *K.T.* como ferramenta de aprendizagem acelerada

Este *software* permite aos formandos/alunos aprender de forma acelerada, através de vários métodos e ajudas, como o enviar perguntas falhadas para o fim da fila, o “modo mental”, entre outras funcionalidades, ou até pelo facto de uma pessoa estar a aprender enquanto joga, que é uma forma natural de aprender, e que leva a que alunos possam estar a competir entre eles para ver quem tem melhores notas, e sem se aperceberem, memorizando comandos, já tendo tido resultados muito satisfatórios em vários alunos e formandos.

Permite assim com 10 minutos de treino obter a prática que necessitaria de várias horas de uso real dos sistemas em causa para aprender.

Por exemplo, alguém pode praticar comandos algo complexos, por exemplo 100 vezes em meia hora, que levaria semanas ou meses de prática real em contexto de trabalho para as adquirir, até porque sejamos francos, ninguém vai fazer backups centenas de vezes numa semana ou mesmo num mês, e pode repetir essas instruções centenas de vezes numa hora a praticar nestes exercícios. E todos sabemos que é pela prática que se assimilam estas coisas.

Eu aposto assim mais na repetição e assimilação, do que em exercícios morosos onde executariam os mesmos comandos pouquíssimas vezes.

O Knowledge Tester para ensino e exames presenciais ou remotos

Com este *software*, pode-se formar/ensinar também, pois podemos incluir ajudas em cada questão, para que o formando possa aprender sobre o porquê de ter falhado uma resposta, e até praticar depois num terminal ao lado, podendo sempre tirar dúvidas com o formador no local.
E dado que permite a criação de exames encriptados, que geram relatórios encriptados e que não poderão ser alterados, e que terão de ser devolvidos ao formador no fim do exame, pode ser usado para a execução de exames com avaliação imediata, sem a possibilidade de falhas de segurança nos mesmos.

A avaliação automática dos exames ajuda na tarefa também, e será melhorada posteriormente.

Tem até métodos de encriptação para a realização de exames remotos, autenticação via *web* para evitar que os exames sejam corridos em duplicado, relatórios, etc.

E mais importante, ele avalia automaticamente as respostas, mesmo podendo ser múltiplas as aceites como correctas, e avalia o examinado em tempo real, poupando trabalho ao formador, e gerando relatórios de tudo.

O *Knowledge Tester* para ensino e exames remotos

Da mesma maneira que no ensino/formação presencial, dado que os exames são encriptados com *AES 256*, que impede aos alunos a alteração do relatório gerado pelo exame, podem os relatórios ser devolvidos ao formador ou professor após o término do exame, sem o risco de serem alterados.

Tem este *software* também a capacidade de enviar a nota do exame bem como estatísticas principais, ao formador, via *web*, que terá sempre controlo sobre como vai cada aluno, e a nota com que terminou, enquanto grava em disco o relatório resumido após a resposta de cada pergunta, para o caso de a máquina do aluno poder crashar entretanto, e poder assim manter a nota guardada no disco.

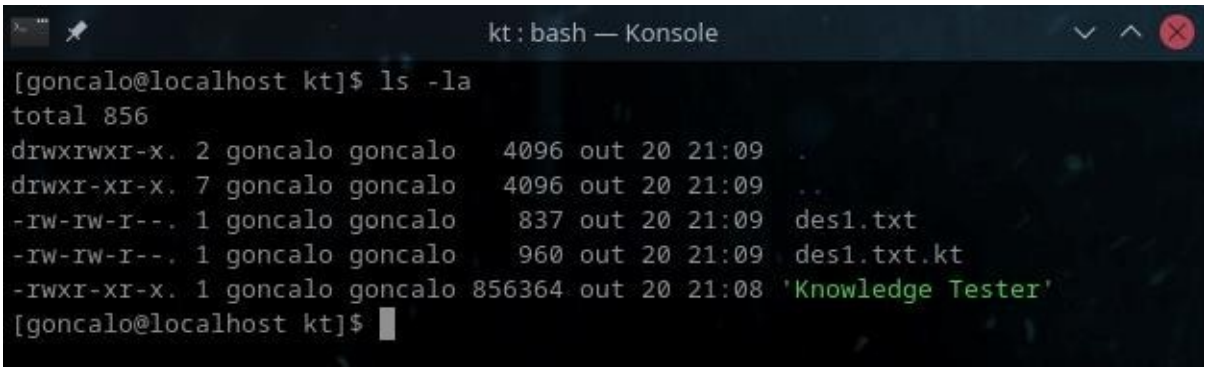
Teoricamente submete automaticamente o relatório também via *web* para o servidor do formador, caso exista, mas essa funcionalidade não tive tempo de aperfeiçoar em 2023, entre muitas outras funcionalidades.

É assim útil no ensino e formação à distância, e até exames à distância, com total segurança.

Instalação

Este *software* não requer qualquer instalação, basta que o utilizador faça *download* do mesmo, algures na página do autor (*www.goncalo.pt*), e mova o ficheiro para uma pasta à escolha, como neste caso do exemplo abaixo, a pasta "*kt*".

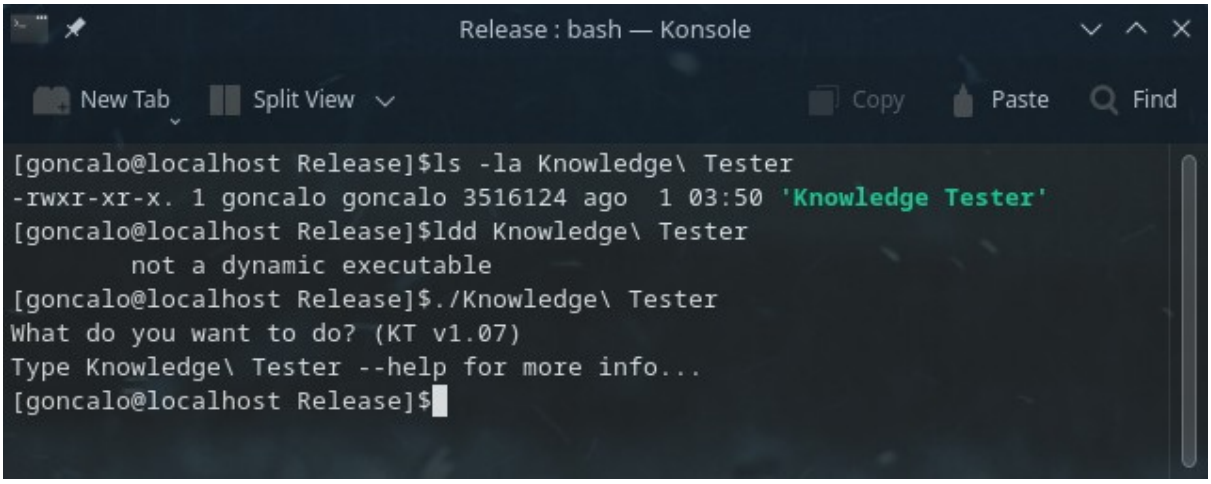
O utilizador apenas tem de executar o comando *chmod* para dar permissões de execução do mesmo ao utilizador, como por exemplo um "*chmod 744 Knowledge\ Tester*" (não esquecendo a barra invertida antes do espaço), e o ficheiro ficará com um tom esverdeado e pronto a executar. Um chmod 744 ou 500 ou 544 ou 700, etc:



É importante referir que o ficheiro não requer que quaisquer librarias sejam instaladas, como

se pode ver abaixo, não tendo dependências, basta sacar o ficheiro, dar permissões de execução, e ele executa.

É por isso que tem um tamanho um bocado maior, é por ter mesmo as librarias essenciais como a *libc* estaticamente incluídas no executável, e de resto ser feito tudo o mais do zero possível:



Ou seja, não é necessária qualquer instalação para usar o *software*, *é* sacar, e usar.

Requisitos mínimos

Os requisitos são mínimos em termos de memória ou *CPU*, apenas temos de ter em conta que foi criado para funcionar em processadores de 64 *bits*, pelo que se correr num de 32 falhará, ou seja, as máquinas virtuais criadas terão de ser de 64 *bits*.

O outro requerimento mínimo é obviamente o sistema operativo, tendo sido criado para correr em qualquer máquina *Linux*, e em princípio funcionará em praticamente todas as actuais, pois foi compilado com librarias estáticas incluídas (daí o seu tamanho maior enquanto executável), para que funcione mesmo em máquinas que não tenham as librarias mínimas já instaladas.

Posso dizer que já criei máquinas virtuais com 25 MB de tamanho compactada (disco a rondar os 25 MB apenas, ou seja 0.025 GB de tamanho), e com apenas uns 50 MB de memória, e a versão do terminal funciona na perfeição.

Para a versão gráfica não poderíamos corrê-la obviamente numa versão core sem interface gráfica, mas já a instalei em distribuições em que o sistema instalado ocuparia apenas uns 200 MB, como a Slax, e funciona na perfeição.

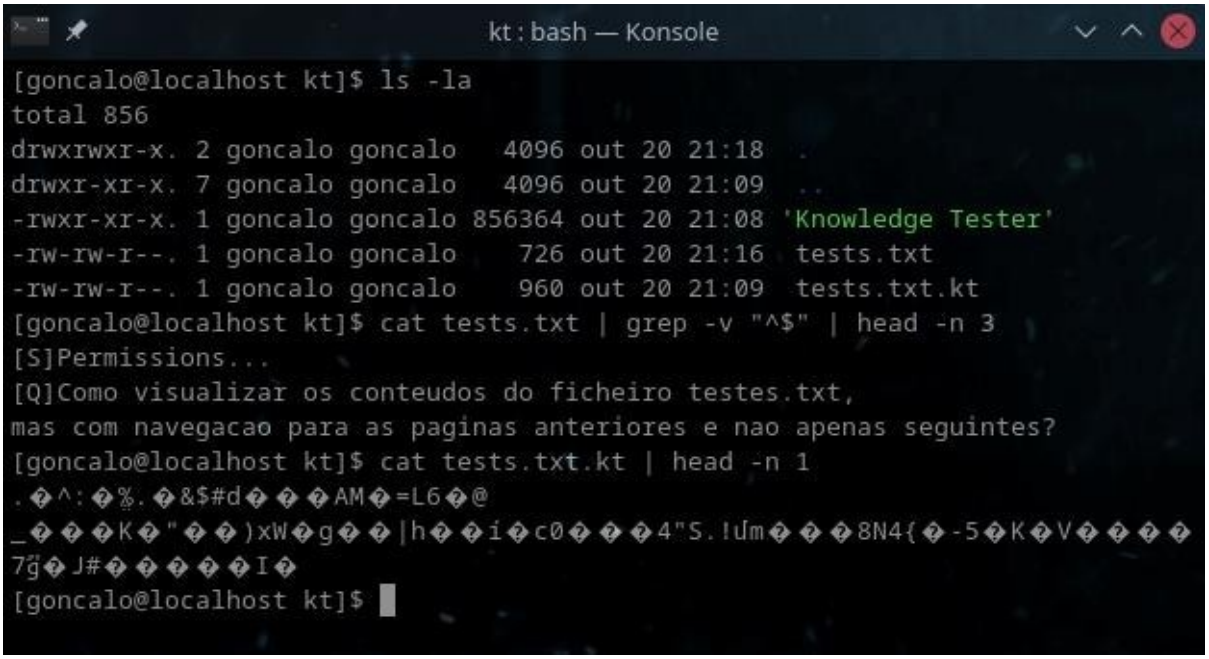
A aplicação ao ser executada em modo de terminal, verifica se é possível criar o ambiente gráfico, e se não for, ela mesma muda para o modo de terminal, pelo que funcionará sempre sem erros, mesmo que não possa aceder a um modo gráfico e nós o possamos pedir.

Tipos de ficheiros do Knowledge Tester

Existem dois tipos de ficheiros principais neste *software* em termos de conteúdo:

- Ficheiros em modo de texto bruto (*raw*);
- Ficheiros em binário (não legíveis);

Os ficheiros de exame em modo de texto, podem ter a extensão que o utilizador desejar, estando no exemplo abaixo com a extensão ".txt" (o ficheiro "*tests.txt*"), enquanto que os ficheiros de exame do *Knowledge Tester* em modo binário, têm a extensão "*.kt*", como o ficheiro "*tests.txt.kt*" abaixo:



Podemos ver acima que o ficheiro de extensão "*.kt*" é ilegível, pois não está em modo de texto *raw*.
Aprenderemos a criar ficheiros "*.kt*" posteriormente.

O Modo Terminal e o Modo Gráfico

Esta aplicação começou originalmente por funcionar apenas em modo gráfico no *Linux*.

Mais tarde, como quis migrar a mesma para *Windows*, e dado que o *Windows* não respeitava os mesmos padrões de terminal do do *Linux*, decidi criar uma *GUI*, uma interface gráfica, para que no *Windows* apenas funcionasse com a mesma.

Ele por defeito tenta executar no modo gráfico, mas caso não existam recursos para tal, ou caso estejamos a executar a aplicação num *Linux* sem modo gráfico, como por exemplo numa micro-distribuição, ou em modo de servidor, ou numa consola sem interface gráfica, ele passa automaticamente para o modo de terminal.

Podemos também invocar a aplicação forçando o modo de terminal, com o parâmetro "*--terminal*", sendo que mesmo que ele tenha requisitos para funcionar em modo gráfico, arrancará no modo de terminal com este parâmetro.

Os Parâmetros

Basta que usemos "--" antes do parâmetro, e ele tomará efeito, como por exemplo "*--terminal*", para a aplicação arrancar no modo de terminal.

Podemos também colocar em frente ao mesmo a opção de 0 ou 1 (Falso ou Verdadeiro), sendo que um "*--terminal 0*" irá forçar o modo de terminal a estar desligado, ou seja, arrancará em modo gráfico.

Existem outros parâmetros que requerem que introduzamos algum texto a seguir, como por exemplo o "*--password*", e neste caso colocar "0" à frente não desligaria, apenas o faria crer que a *password* seria "0".

Usando estes dois parâmetros seria algo como "*--terminal 0 --password PalavraPasse --mentalMode*". Neste exemplo, o modo terminal arrancaria desligado (arrancaria em modo gráfico), com a *password* "PalavraPasse", e em modo "*--mentalMode*", pois se não tiver "0" ou "1" a seguir, arranca por defeito com "1", ou seja, ligado.

Este tipo de parâmetros obedece às regras do *Linux*, em que usamos por norma um hífen para definir parâmetros de uma só letra, e dois para definir parâmetros de mais que uma letra.

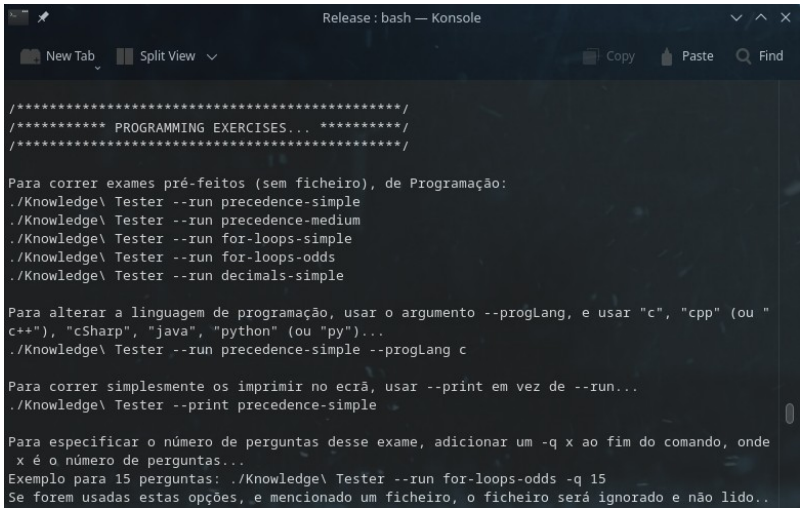
Por exemplo, um "*-la*" seria o mesmo que usar "*-l -a*", enquanto que um "*--la*" significaria um único parâmetro de nome "*la*".

O ter uma maiúscula no começo de cada palavra do parâmetro veio do meu hábito de usar *camelCase*, que é uma norma de separar várias palavras no nome de uma variável sem colocar traços no meio.

Invocar a Ajuda

No *Knowledge Tester*, podemos invocar uma ajuda rápida, com o parâmetro *--help*.

Escrevemos assim "*./Knowledge\ Tester --help*", ou "*./KT --help*", e surgirá um texto enorme com um conjunto de instruções que podemos usar, e como as usar:



O ficheiro *.KTargs.cfg*

Com este ficheiro, podemos fazer com que a aplicação arranque por defeito com certos parâmetros.

Assim, se por exemplo quisermos sempre correr a aplicação no modo de terminal, basta que coloquemos nesse ficheiro o texto "*--terminal*", e sempre que a aplicação correr, ela arrancará no modo de terminal.

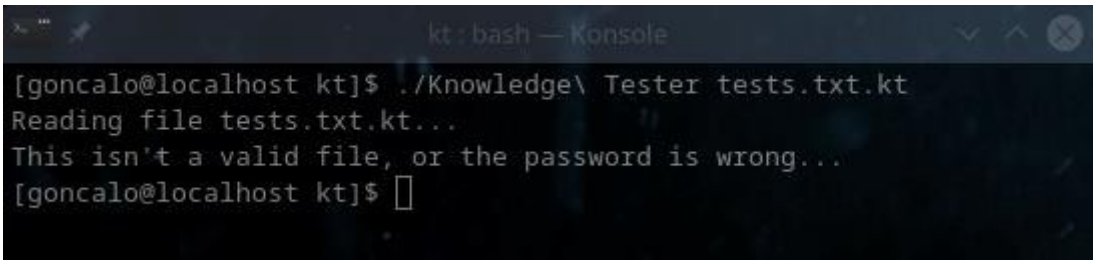
Podemos usar o ficheiro para outros parâmetros também.

Mas atenção que os parâmetros definidos manualmente, se irão sobrepor. Ou seja, se o ficheiro contiver um "*--terminal 0*" que forçaria o arranque do modo gráfico (ou seja, modo de terminal desligado), mas arrancarmos o comando com "*./KT --terminal*", ele arrancará em modo de terminal, pois apesar de no ficheiro de configurações ele ter que é para arrancar em modo gráfico, como eu forcei o "*--terminal*", que é o mesmo que "*--terminal* 1", ele arranca no modo de terminal na mesma, pois se sobrepõe à configuração no ficheiro.

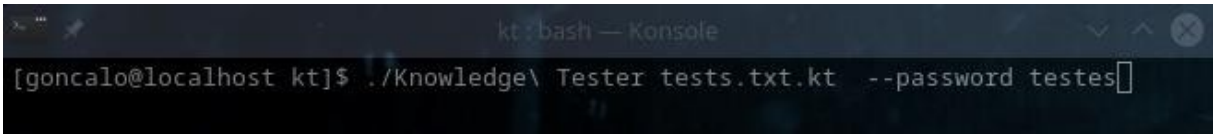
Como executar um exercício ou exame

Se corrermos a aplicação sem nada mais, teremos apenas um alerta de que podemos usar o argumento "*--help*" para obter ajuda sobre como o usar, pois temos de ter no mínimo um argumento, que é o do ficheiro a ser aberto.
Assim, desde que tenhamos um ficheiro de exercícios ou exames criado, só temos de correr a aplicação seguida do nome do dito ficheiro, e a aplicação irá abri-lo de imediato:

Se surgir um erro de "*password is wrong*", é sinal de que o exame está protegido com palavra-passe, e só abrirá se usarmos o argumento "*--password*" seguido da mesma:



Abaixo podemos ver como abrir um ficheiro de testes, com a *password* fornecida pelo autor do mesmo:

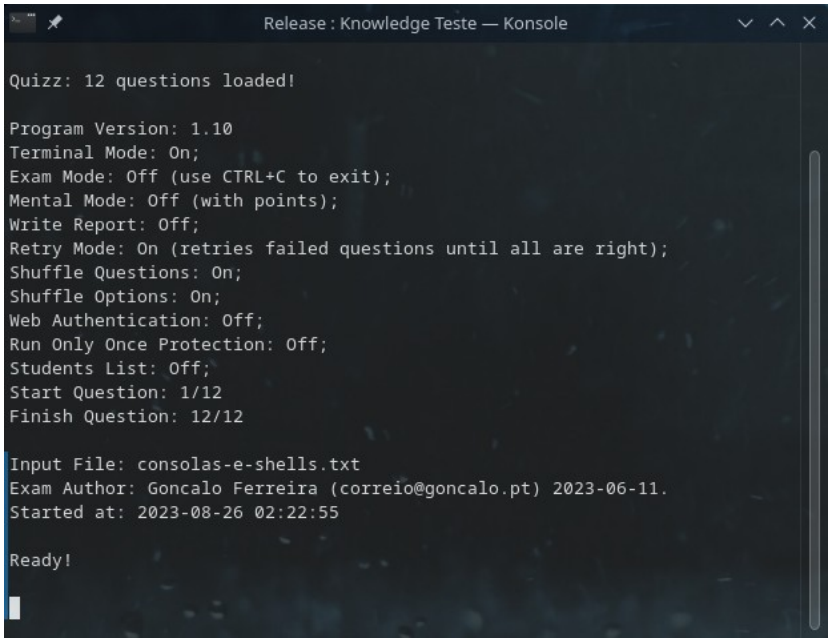


Após ser aberto um exame, surge-nos um ecrã de abertura, onde é exibida (por tradição) a lista de perguntas e respostas, onde podemos ver que algumas têm algo como "[1][0][0][0]" (uma opção certa "[1]" e várias erradas "[0]", nas de escolha múltipla), ou "[W]", que representa uma resposta directa e não de escolha múltipla.

Abaixo vemos as mesmas num arranque em modo “*exam mode*”:

Quando no modo “*exam mode*”, todas as respostas são escondidas, como podemos ver acima, aparecendo todas como “[0]”, enquanto que no modo normal, aparecem visíveis:

Acima podemos ver também que além de nos ser dito quantas questões compõem o exame, é-nos mostrada uma tabela pequena com as opções do exame, como:



Vamos analisar cada uma delas.

Program Version

Este campo define a versão do software necessária para correr o exame, sendo que se por exemplo o exame tiver definido com uma versão 1.02, e o software for apenas a versão 1.01, o exame não correrá.

Terminal Mode

Aqui poderemos ver um “*On*” or “*Off*”, e diz-nos simplesmente se estamos no modo de terminal ou no modo gráfico.

Exam Mode

Este é o modo de exame, e quando activo, não nos permite sair pressionando o atalho "*CTRL+C*", para evitar que um formando/aluno pressionem "*CTRL+C*" quando vêem que a resposta é esse mesmo atalho, e o usem ao invés de escolher o número correspondente (já aconteceu).

Deste modo, durante o modo de exame, só se consegue sair do programa, escrevendo duas linhas com "*:q!*" nas respostas directas, e ao fim da segunda linha assim, e pressionando *Enter*, o programa termina.

Mental Mode

Este é um modo de prática rápido, em que o utilizador visualiza na sua cabeça a resposta, e vê se acertou, premindo uma tecla, ao invés de responder para obter pontuação, e será falado mais abaixo.

Write Report

Este argumento terá um "*On*" ou "*Off*", e define se será gravado um relatório no fim do exame, o que está activado por defeito quando no modo de exame, mas poderemos desactivá-lo assim nesse modo ou activá-lo noutros modos em que costuma estar desligado por defeito.

Não me recordo ao escrever este tutorial se desliga automaticamente, caso esteja "*Off*" o relatório que é guardado entre cada resposta respondida, algo que terá de ser confirmado mais tarde, e que é um relatório que evita que caso o computador possa "*crashar*" a avaliação do exame se perca.

Retry Mode

Neste modo, caso esteja "*On*", as perguntas falhadas pelo utilizador, são reenviadas para o fim da lista, pelo que assim, o utilizador irá repetir as perguntas após chegar ao fim do exame, e o mesmo só terminará até ele ter acertado em todas elas.

O objectivo é a pessoa repetir as perguntas as vezes que forem precisas, até as responder todas bem, ou seja, até as ter a todas memorizadas.

Shuffle Questions

Com este modo activo (com "*On*"), as perguntas são baralhadas, e assim aparecerão sempre em ordem diferente em cada exame, para evitar que o utilizador sem querer memorize as respostas visualmente, ao invés de aprender a matéria.

Shuffle Options

Com este modo activo (com "*On*"), as opções de cada pergunta, quando forem de resposta múltipla, serão baralhadas sempre que a pergunta aparece, para evitar que o utilizador memorize a resposta visualmente ao invés de aprender a matéria.

Web Authentication

Esta opção indica-nos, caso esteja "*On*", que existirá autenticação via *web*, e que teremos de usar parâmetros como o "*--webAuthUser*" para indicarmos à aplicação qual é o nosso utilizador, e através dele teremos permissão ou não, via *web* (contactando o servidor), sobre se poderemos começar o exame em questão ou não.

Run Only Once Protection

Com esta protecção ligada, o exame em questão só corre uma única vez, e se tentarmos correr o mesmo exame novamente nessa máquina, não conseguiremos com as protecções activadas, que não são muito avançadas de momento, mas o suficiente para evitar que alunos/formandos tentem correr o mesmo exame duas vezes sem que o formador note.

Esta opção tornou-se obsoleta a partir do momento em que criei a opção de *web authentication*, pois com a "*Run Only Once Protection*", havia ainda maneira de se iniciar o exame, remotamente, de forma a correr em várias máquinas diferentes, enquanto que com a autenticação via *web*, assim que o exame pede autorização ao servidor para iniciar, perde a mesma, e a 2ª tentativa de autenticação falhará logo.

Students List

Esta opção permite-nos, ao correr um exercício, ver uma lista de estudantes aparecer, de forma aleatória (um por pergunta), para que seja atribuída uma pergunta de forma aleatória, a cada estudante, muito útil para aulas ao vivo e testar alunos.

Para isso, ao correrm os a aplicação, definimos qual o ficheiro com a lista de estudantes, e ele vai fazer *shuffle* aos nomes, e colocar um por pergunta.

Quando chegamos ao fim da lista, ela é reiniciada.

Start Question & Finish Question

Aqui surgem a primeira questão e a última questão do exame/questionário a serem lidas pelo programa, sendo que todas as anteriores ou seguintes não surgirão no exame.

Estas opções de começo e fim, permitem a que um utilizador possa por exemplo fazer um teste com as perguntas 10 a 20, de um questionário com 100 perguntas, para escolher quais quer treinar, para poder aprender mais rápido. Desta maneira, pode repeti-las até as aprender (ou memorizar), em blocos de dez de cada vez, e acelerar o processo, ou mesmo 3 de cada vez, 5, etc.

I*nput File*

Aqui aparece obviamente qual é o ficheiro de testes que é lido no exame que está a decorrer.

Exam Author

Neste campo, temos apenas o nome, e talvez contacto, de quem criou o exame ou exercício em questão.

Started At

Aqui surge a data correcta de começo do exame, sendo que é importante que o utilizador defina a hora correcta do sistema, para que a data de começo e fim apareça correcta no relatório, com um simples *date s "202110-20 22:10:00"*.

E após o "*Ready!*" final do ecrã de abertura, só temos de pressionar uma tecla, e o exame começa.

O Modo Gráfico

O modo gráfico foi desenvolvido como complemento ao modo de terminal, e para criar uma interface que fosse transversal a todos os sistemas operativos (para migração posterior para *Windows*, *MacOS*, etc.

Tem inúmeras funcionalidades, e abordaremos algumas de seguida.

Mas antes de começarmos, convém mostrar uma muito importante: o atalho de ajuda.

Ajuda

Em qualquer momento, usando o atalho "*CTRL+H*" ou "*F1*", podemos invocar este pequeno painel com uma lista de atalhos que podemos usar no *software*, durante o modo gráfico:

Scanlines

As *Scanlines* são uma funcionalidade que desenvolvi inicialmente para o meu *Game Engine* em *C++*, e que apliquei ao *software* de Formação, e que no fundo simula o efeito dos computadores ligados às televisões *CRTs* antigas, e que é representado pelas riscas horizontais entre linhas, como podem ver abaixo:

Estas *Scanlines* podem ser activadas com o atalho "*CTRL+S*", e desactivada com o mesmo atalho, e podemos aumentar ou reduzir a sua intensidade com os atalhos "*F9*" e "*F10*".

Temas de Cores

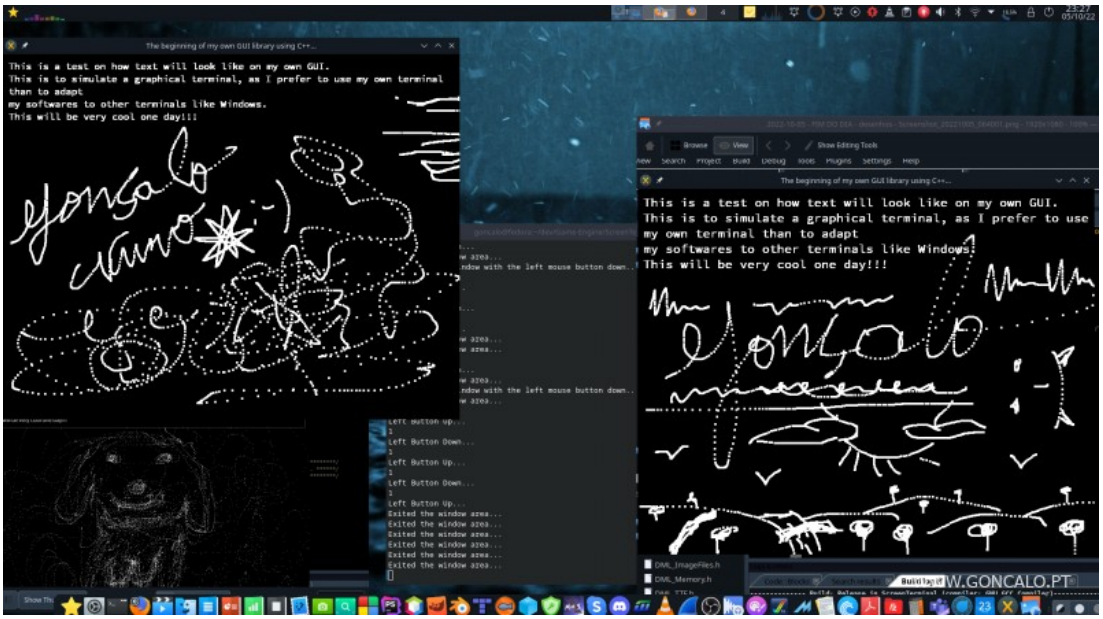
Este *software* tem vários temas predefinidos de cores, e esta ideia nasceu em parte porque eu por defeito usava o tema habitual de fundo negro e letras brancas, mas tive formandos que me referiam que preferiam ter letras negras sobre fundo branco.

Nessa altura decidi criar vários temas predefinidos, que mudam praticamente todas as cores, tendo um de fundo negro, outro de fundo branco, um tema verde fósforo, um verde escuro, um azul, um vermelho, e talvez mais alguns.

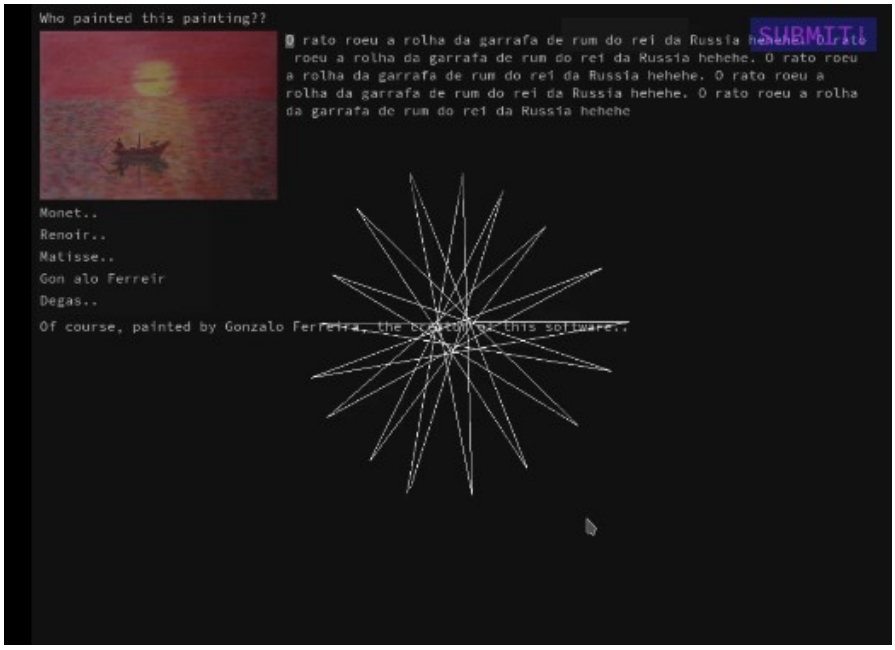
Os atalhos para mudarmos de tema são as teclas "*F11*" para o tema anterior, e "*F12*" para mudarmos para o tema seguinte, e só experimentando é que teremos ideia de como funciona.

Desenhar com o rato

O *software* permite-nos ao pressionar a tecla "*CTRL*", enquanto pressionamos o botão esquerdo do rato, desenhar linhas no ecrã, normalmente usado para pequenas anotações quando queiramos tirar um *screenshot*, ou *print-screen*, como eu lhe costumo chamar:



Se pressionarmos a tecla "*Shift*" e "*CTRL*" em simultâneo, poderemos desenhar linhas rectas:



Isto pode ser usado talvez para sublinhar linhas onde possamos ver um erro num exame, para depois tirarmos um *screenshot* para enviar ao formador.

Screenshots

Este *software* permite-nos a qualquer momento, através de um atalho, tirar um *screenshot* à janela actual do *software*, gravando assim de forma automática no nosso disco, uma imagem do que estamos a ver no ecrã.

Vejamos o que acontece, através da imagem abaixo, quando pressionamos o atalho "*CTRL+P*":

Podemos ver na imagem acima, que no canto inferior direito do ecrã, nos surge uma mensagem do género:

Esta notificação acima aparece quando tiramos um *screenshot* à janela da aplicação, e neste caso, devido a termos usado o atalho "*CTRL+P*", foi no formato de *PNG*.

Atalhos de *Screenshots* disponíveis

Os atalhos disponíveis são:

CTRL+P – Gravação de um *screenshot* em formato *PNG* de 16 milhões de cores (*RGBA*);

CTRL+O - Gravação de um *screenshot* em formato *BMP*;

CTRL+I – Gravação de um vídeo em formato *GIF* (mas esta encontra-se desactivada). Quando pressionamos *CTRL+I*, ele começa a gravar todos os frames, todos os nossos movimentos e alterações no ecrã, e quando voltamos a pressionar *CTRL+I*, ele pára de os gravar, e ficaria pausado por uns segundos enquanto fazia a quantização de cores e guardava a imagem do *GIF* animado. Desta forma, tínhamos um *GIF* animado com animações feitas por nós. Deixei desactivada devido a possíveis erros de *segmentation fault* quando os vídeos (*GIFs*) eram muito grandes, que não quis perder tempo a corrigir, por falta de tempo, fica para outra altura;

CTRL+U - Gravação de um *screenshot* em formato *PNG* de 256 cores (indexada);

Outros atalhos e *features*

Aqui vou deixar algumas notas que por falta de tempo em procurar melhor sítio do manual para as colocar (deixarei para 2025 talvez, quando voltar a mexer neste *software*), deixo aqui para que não fiquem esquecidas.

Submeter perguntas em resposta directa

Antes, no modo de terminal, para submetermos uma resposta em questões de resposta directa (resposta escrita), tínhamos de com a tecla *Return/Enter*, submeter 2 ou 3 linhas vazias, para ele submeter a resposta.

Mas devido a queixas de quem por vezes (sem eu compreender como), deixavam estar a tecla pressionada vários segundos e submetiam supostamente as respostas sem querer, esse problema ficou resolvido, pois agora nos modos de Exame (no *Exam Mode*), só se podem submeter respostas escritas ou com o "*CTRL+Enter*" no modo de terminal, ou com tanto o "*CTRL+Enter*" como o botão "*Submit*" com o rato.

Com ambos, podemos submeter a resposta directamente tanto no modo de exame como nos outros.

De resto, se não estivermos no modo de exame, podemos submeter as respostas com várias linhas vazias no fim da resposta, clicando várias vezes na tecla "*Enter*".

Isto melhorou o antigo sistema, que era algo chato, de obrigar os utilizadores a pressionar várias vezes a tecla "*Enter*" até a resposta ser submetida.

Atalho de saída *CTRL+W* não funciona no modo de exame

O atalho de saída do programa, o "*CTRL+W*", que antes era usado para se sair do programa no modo gráfico, não funciona quando estamos no Modo de Exame, para evitar que alguém saia do exame sem querer.

Da mesma maneira, não podemos sair também no modo de terminal com o atalho "*CTRL+C*" durante um exame (em Modo de Exame), para evitar que alguém pressione o tal atalho sem querer. Pois por estranho que pareça, há quem veja uma resposta múltipla com um "*CTRL+C*" e em vez de seleccionar, pressione mesmo o atalho "*CTRL+W*" saíndo do exame, algo que agora não haverá risco de acontecer mais.

Modo de *Insert* por defeito no Editor de Texto

No Editor de Texto, arrancamos por defeito com o modo de *Insert* activado, e para o desligarmos, teremos simplesmente de pressionar a tecla *Insert*.

Os atalhos de Ajuda

Agora tanto funcionam os atalhos "*F1*" como o "*CTRL+H*" para visualizarmos o ecrã de ajuda, quando estamos no modo gráfico.

Devemos depois voltar a pressionar um destes atalhos para sair do tal modo de ajuda.

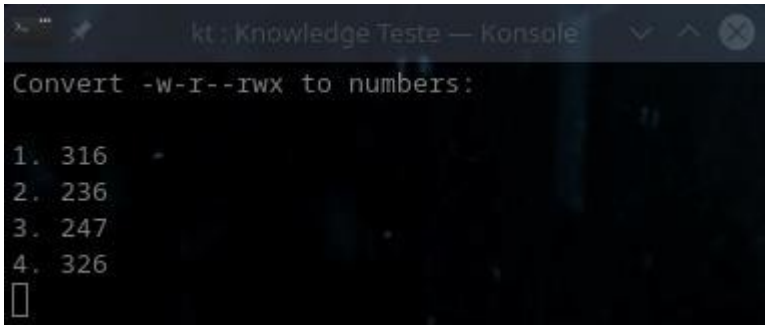
Os modos de execução do *software*

Aqui serão explicados os vários modos de execução deste *software*, como o modo *Mental Mode*, o *Exam Mode*, etc.

Normal Mode

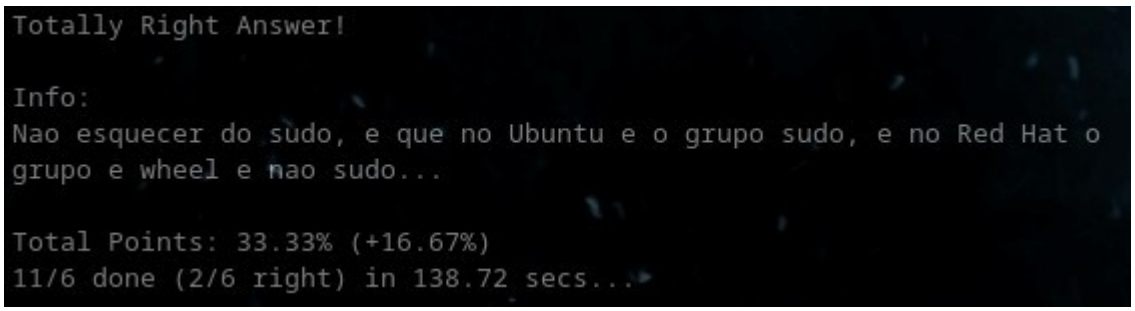
O modo normal é o modo que temos quando não invocamos o argumento "*--examMode*" nem o argumento "*--mentalMode*" ao correr o programa.

No modo normal, tal como no modo de exame, temos um número atribuído a cada resposta, no caso de escolha múltipla, para que ao pressionarmos a tecla desse número, ele nos confirme se acertámos ou falhámos a resposta:



E é semelhante ao modo de exame, no sentido de que temos uma pontuação, mas com algumas diferenças.

Uma diferença importante ao modo de exame, é que aqui temos as ajudas visíveis, que são os textos definidos com a *tag* "[I]" na criação do exame, e que visam dar uma ajuda ao utilizador sempre que certa ou falha uma resposta, para que ele possa aprender:



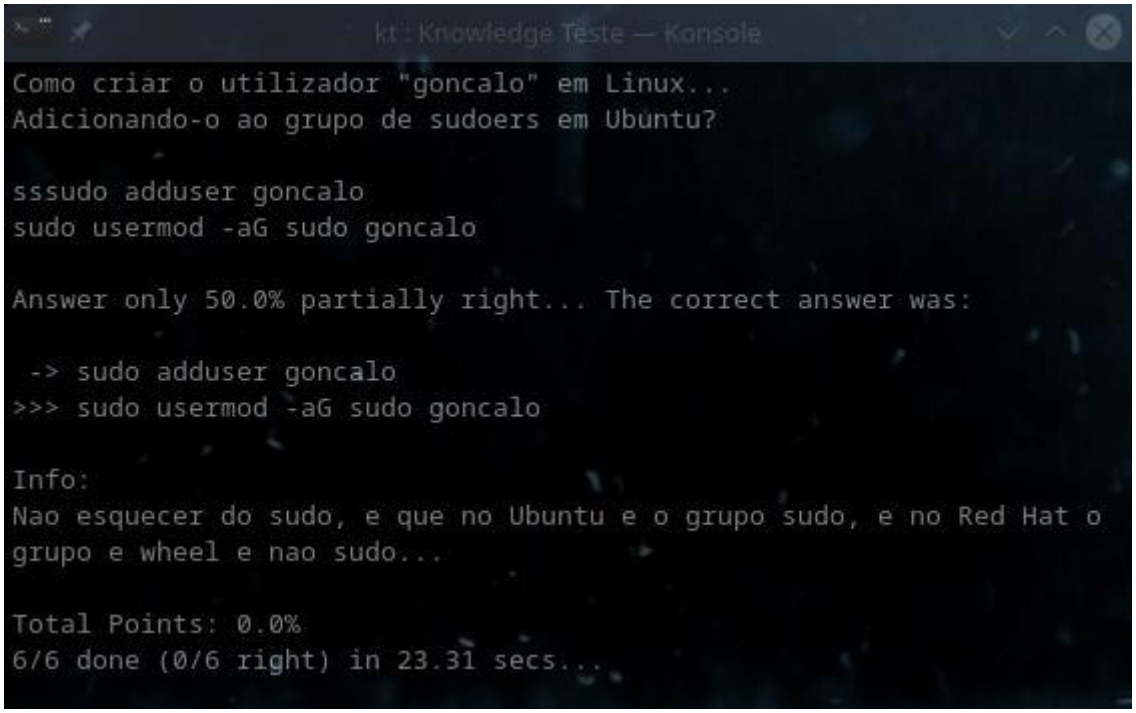
Podemos ver na imagem acima, que após o título "*Info:*", surge um texto de ajuda ao utilizador, gerado pelo criador do exame. Esta funcionalidade não existe obviamente no modo de exame, ou o utilizador saberia sempre as respostas.

Outra delas, é a de que podemos cancelar o exame a qualquer momento com o atalho "*CTRL+C*", coisa que não acontece no modo de exame, onde tal atalho foi cortado por ter visto um caso de alguém que pressionou "*CTRL+C*" para responder a uma questão em que "*CTRL+C*" era a resposta, ao invés de escolher o número da mesma.

Claro que isto apenas funciona no modo de terminal, pois no modo gráfico, o atalho "*CTRL+C*" não funcionará, apenas o atalho "*CTRL+W*", mas só se estivermos fora do Modo de Exame.

Dentro do Modo de Exame, para saírmos, temos de escrever duas linhas seguidas com "*:q!*" em cada uma, antes do pressionar da tecla "*Enter*" final, em memória ao velhinho *VIM* no *Linux* (onde é o comando para se sair sem alterações).

Outra diferença, é a de que se uma pergunta de multi-linhas for respondida apenas parcialmente correcta, no modo de exame ela não voltaria para o fim da lista, e neste modo volta, e o utilizador terá de a repetir as vezes que forem precisas até estar 100% certa, para o teste terminar.

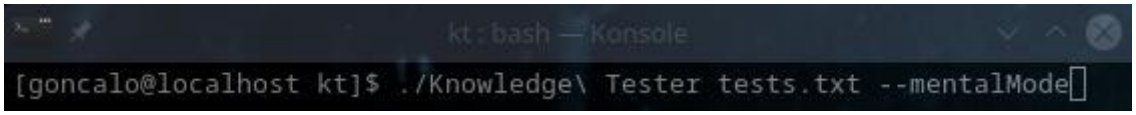


E a principal é a de que no modo exame, se a resposta não for vazia, não poderemos responder mais à mesma, enquanto que aqui, se não usarmos o argumento "*--retryOff*", as perguntas serão enviadas para o fim da lista para as tentarmos novamente, mil e uma vezes, até as acertarmos a 100%.

Mental Mode

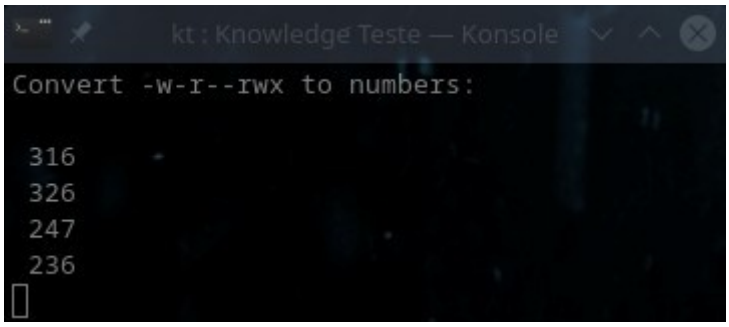
Este é um modo de prática rápido, em que o utilizador visualiza na sua cabeça a resposta, e vê se acertou, premindo uma tecla, ao invés de responder para obter pontuação, e nela não só não existe pontuação como não existem teclas a pressionar, basta um simples pressionar da tecla *Enter* (ou qualquer outra), e vemos a resposta, e com outro pressionar, vamos à resposta seguinte.

Ele é invocado ao adicionarmos "*--mentalMode*" à linha de comandos que invoca o exame:



Este modo, permite memorizar muito rápido a matéria, pois sem repararmos, fica tudo gravado nas nossas mentes com as repetições, sem esforço, de forma a que se consigam aprender uma quantidade enorme de comandos com um esforço mínimo, sem cansaço (se o utilizador não abusar dos exames, claro está).

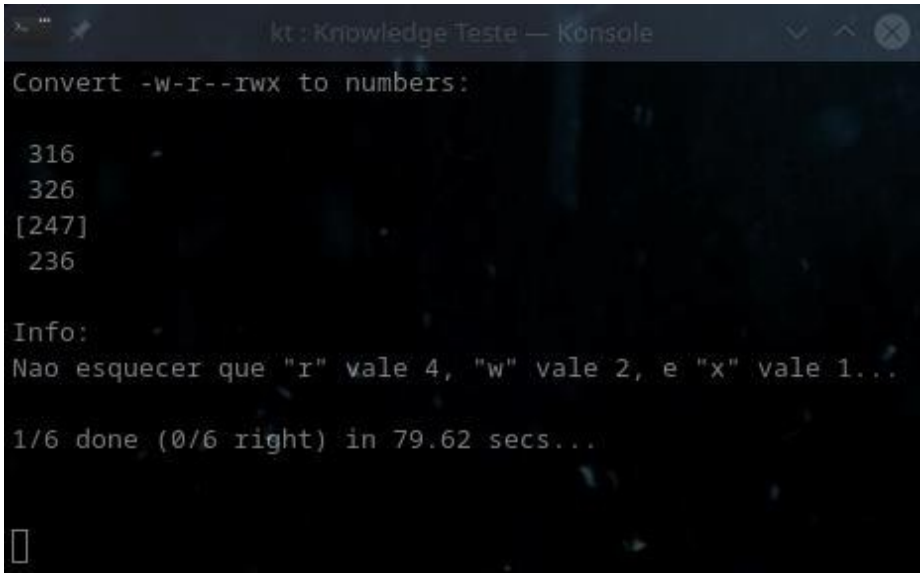
O objectivo é o utilizador "jogar" enquanto tem vontade, de vez em quando, e sem querer ir aprendendo a matéria, e lendo explicações, e depois praticar no terminal.
No modo mental, não existem opções a escolher:



Como se pode ver acima, não há opções para escolher, nem teclas para pressionar, devemos mentalmente pensar qual é a resposta, e pressionar uma tecla qualquer (como por exemplo Enter), e a resposta surgirá, e teremos confirmação mental se acertámos ou não.
Isto torna tudo mais rápido, conseguimos responder talvez ao dobro das perguntas na mesma quantidade de tempo, mas tem um senão: como não escolhemos uma resposta por teclas, o programa não sabe se acertámos ou errámos, e por isso não passa as que falhámos para o fim da lista para as repetirmos mais tarde.

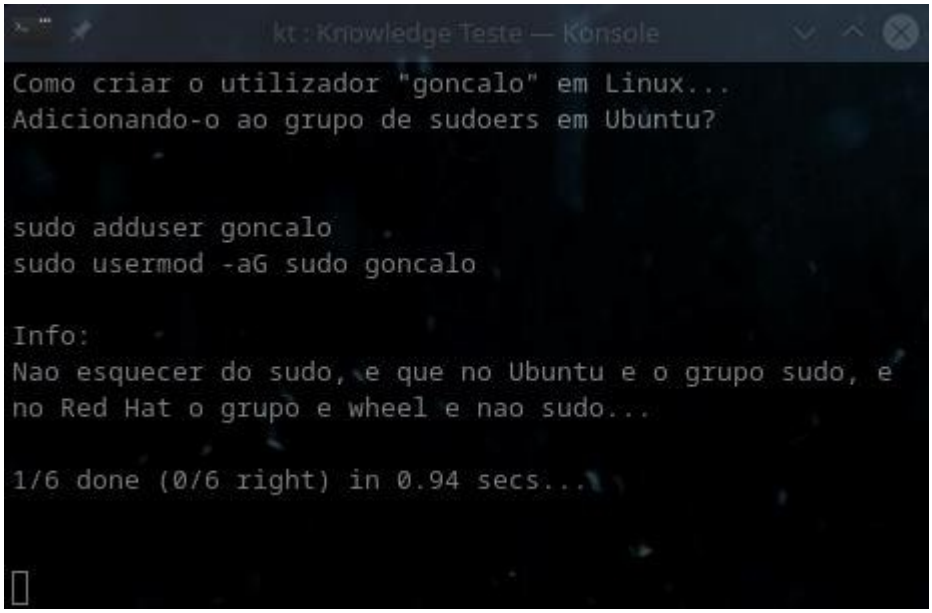
Este modo é especialmente útil se estivermos a estudar para certificações, ou se tivermos muito pouco tempo para aprender algo.

Abaixo podemos ver o que acontece se pressionarmos uma tecla neste modo:



Como podemos ver acima, ao pressionar uma tecla, surgem-nos parêntisis rectos a envolver a resposta certa, para sabermos mentalmente se acertámos na resposta certa ou não.

Também em respostas directas, as mesmas nos aparecem no ecrã com um simples pressionar de qualquer tecla:



Não existe é pontuação, mas é muito bom modo para praticar.

Exam Mode

Este é o modo de exame, e é usado para exames, e tem várias diferenças do modo normal.

Primeiro, só podemos tentar cada pergunta uma única vez, e se a falharmos, não poderemos voltar a tentá-la.

Podemos contudo, pressionar *Enter* numa pergunta (não a responder), e essa pergunta vai para o fim da lista, para a tentarmos mais tarde. Isto é útil para o utilizador deixar para o fim do exame as perguntas mais difíceis que não se lembra, e tentar as que conhece melhor primeiro.

Assim, o utilizador despacha as que sabe, e deixa o tempo do fim para as que não sabe.

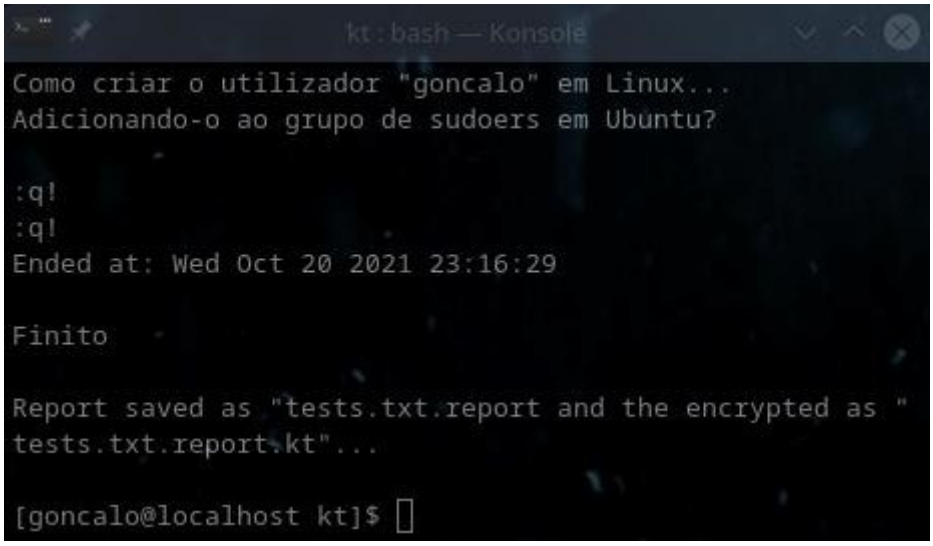
Mas ao contrário do modo normal, aqui o número de perguntas tentadas não sobe para o fim sempre que passamos uma resposta para o fim.

Se o utilizador tentar responder e falhar, não pode voltar a tentar essa questão.

Esta funcionalidade de deixar para o fim questões não respondidas, pode ser anulada com o argumento "*--retryOff*" quando se invoca o exame.

Também, quando activo, este exame não nos permite sair pressionando o atalho "*CTRL+C*", para evitar que um formando/aluno pressionem "*CTRL+C*" quando vêem que a resposta é esse mesmo atalho, e o usem ao invés de escolher o número correspondente (já aconteceu).

Deste modo, durante o modo de exame, só se consegue sair do programa, escrevendo duas linhas com "*:q!*" nas respostas directas, e ao fim da segunda linha assim, e pressionando *Enter*, o programa termina:



Como podemos ver acima, ao escrevermos duas vezes "*:q!*", uma por linha, o exame termina, apesar de ser uma funcionalidade ainda não muito bem testada, e é temporária. Para quem tiver curiosidade, este comando é o mesmo usado no editor de texto *VI/VIM*.

Os relatórios de exames

Sempre que termina um exame, são gerados relatórios do mesmo, um desencriptado, em modo de texto puro (*raw*) que o utilizador pode ler e rever, com extensão extra "*.report*", para ver a pontuação que obteve e onde falhou, e um outro encriptado, de extensão "*.report.kt*", sendo que este outro, só pode ser desencriptado pelo autor do exame, com uma *password* que não costuma ser partilhada com os formandos/alunos, sendo que o objectivo é impedir que seja adulterado.

Por norma, são os dois enviados ao examinador, para verificar a pontuação automática, e adicionar pontos extra aos exames, para os casos de quem se enganou por erros de escrita mas que tinha a lógica certa, ou outros pequenos erros, sendo que por norma, o resultado é sempre acima do dado pelo programa.

O parâmetro "*--writeReport*"

Este parâmetro faz com que a geração de relatórios possa ser desligada no modo de exame (onde normalmente está activada, por defeito), ou activá-la noutros modos que não o modo de exame (onde por defeito está desactivada).

Basta adicionarmos então um "*--writeReport*" à linha de comandos que invoca o *Knowledge Tester* para executar um exame, e eles passam a estar activados, ou então um "*--writeReport 0*" ou "*--writeReport false*" à frente de uma linha de comandos que invoque um exame em modo "*--examMode*" para estarem desactivados, mas claro que só funcionará caso não esteja esse parâmetro forçado (algo que será falado noutra parte deste manual).

Os relatórios têm registado no fim a versão do *software* usado para a execução dos mesmos.

Envio de relatórios via *web*

Quando o exame tem autenticação via *web*, o *software* tenta enviar automaticamente o relatório para o servidor definido pelo examinador, mas esta funcionalidade ainda falha por vezes por isso o ideal é pedir para os examinados enviarem por email os relatórios, pois não vou trabalhar mais nela até 2025.

Relatórios temporários entre perguntas

Existe um relatório que é gravado de forma automática após a resposta de cada pergunta, para o caso de se a máquina do examinado "crashar", haja um relatório que nos diga onde o examinado ia e que nota tinha. Caso haja uma *password* definida, esse relatório entre perguntas penso que não pedirá *password*, ou seja, vai desencriptado (penso eu, verificarei em 2025 quando voltar a pegar no projecto).

Como criar um exame

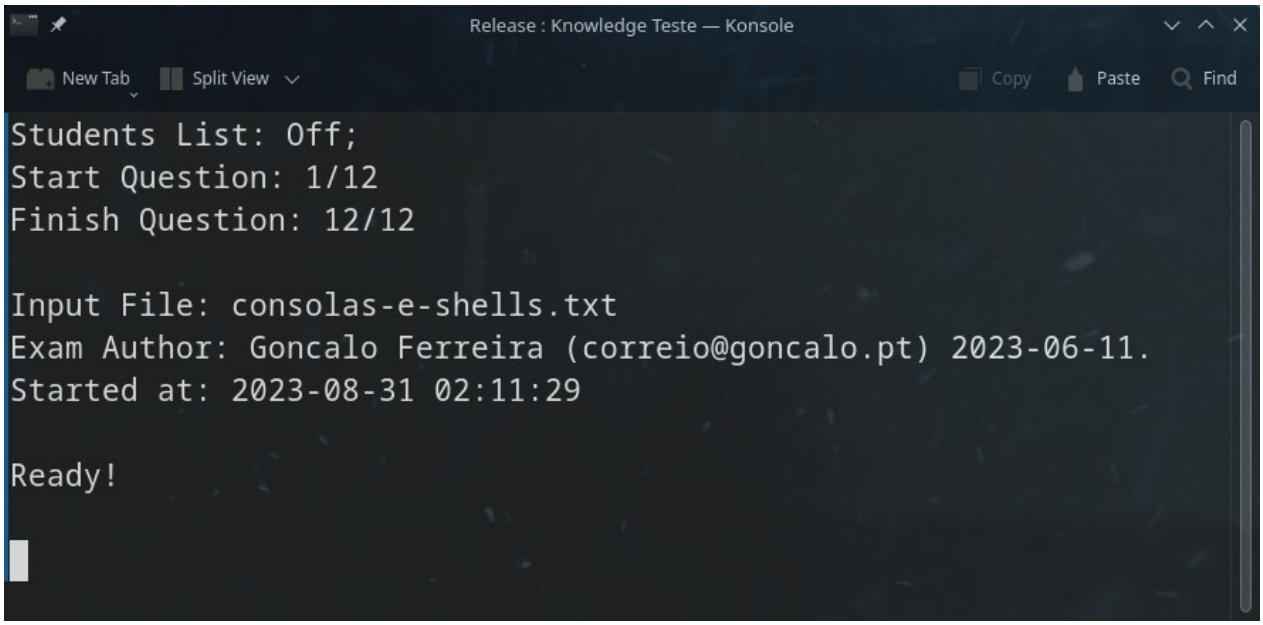
Nos tópicos abaixo, que falam sobre os tipos de perguntas existentes no *software* "*Knowledge Tester*", poderão ver como criar exames.

Não é complicado, só têm de usar algumas *tags* para definir perguntas, respostas, secções, e outras coisas, e são explicadas com mais pormenor nos tópicos seguintes.

Mas antes deixa-se um pequeno resumo das *tags* que se podem usar:

[A] – Author

Esta *tag* é usada para definir o nome do autor do exame, que aparecerá no ecrã de apresentação do mesmo, a seguir a "*Exam Author*":



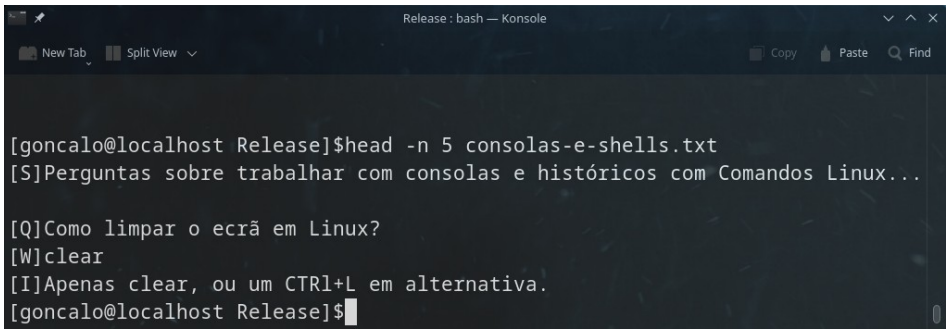
[S] – Section

Esta *tag* é usada para definir uma secção, que de momento não é muito usada, mas que um dia será usada para permitir a que um ficheiro possa ter muitas secções distintas, e que podemos correr perguntas apenas de uma específica, e define o nome de cada secção.

Um dia, criarei menus para seleccionar quais secções de cada ficheiro praticar…

[Q] – Question

Aqui definimos uma questão, ou seja, o texto que aparece para cada pergunta em si:

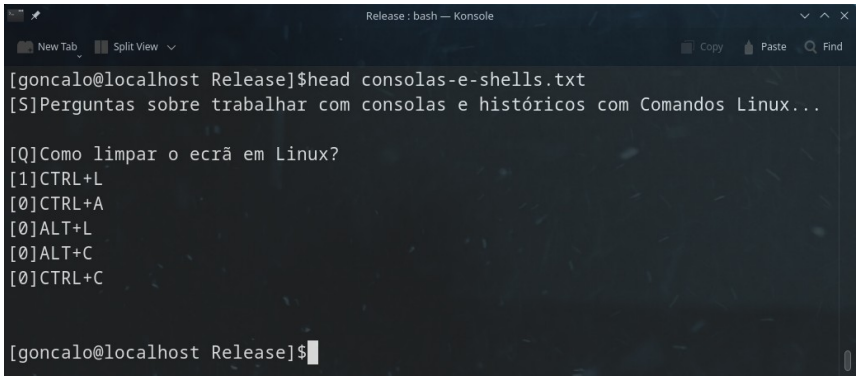


[1] – Right Answer

Esta *tag* define uma resposta certa, pelo que se uma linha tiver esta *tag* antes, será a resposta certa, quando usamos perguntas “de cruzinhas”, onde surgem várias erradas e uma certa e temos de escolher a certa. De momento o *software* não permite escolha múltipla, pelo que só uma poderá ser usada por questão;

[0] – Wrong Answer

Com esta *tag* definimos as respostas erradas, ou seja, todas as respostas que as tenham, serão erradas e farão o utilizador perder a oportunidade de ganhar pontos se as escolherem:

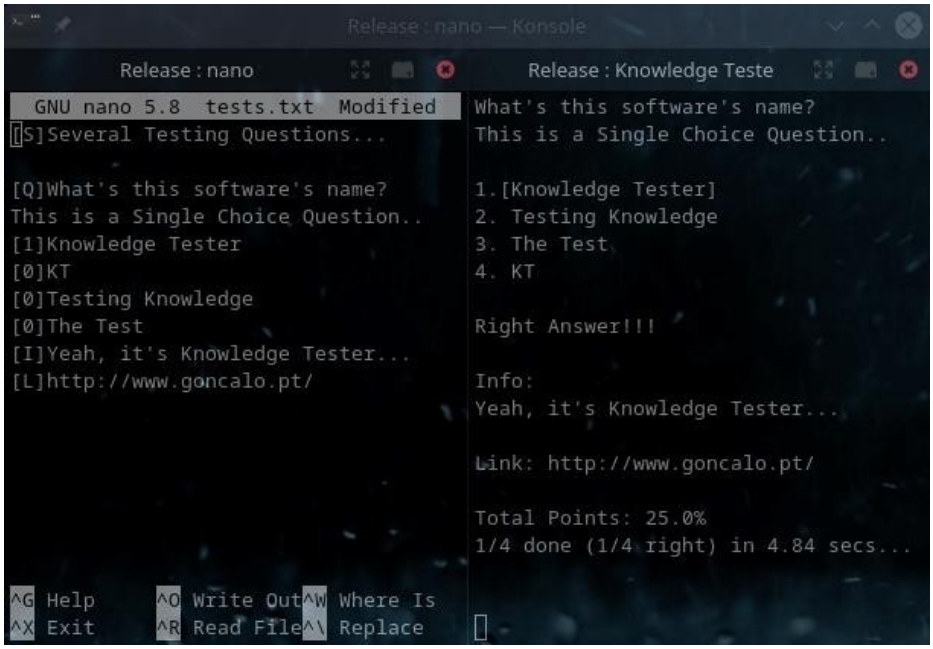


Podemos colocar sempre em 1º lugar a *tag* "*[1]*", porque ele apresenta as opções de forma aleatória por defeito, e por isso não tem problema.

NOTA: É de notar que neste tipo de respostas, seja de escolha por cruzes (uma *tag* e várias ***[0]***), seja em respostas de verdadeiro ou falso (quando temos apenas duas respostas, a verdadeira com "***[1]* –** Verdadeiro", e a falsa com "***[0]*** – Falso"), basta-nos escolher a resposta pressionando as teclas de 1 a 9, pois temos um limite de momento de 9 opções possíveis, e ao ser pressionada a tecla, a resposta é automaticamente enviada, por isso há que ter cuidado ao responder.

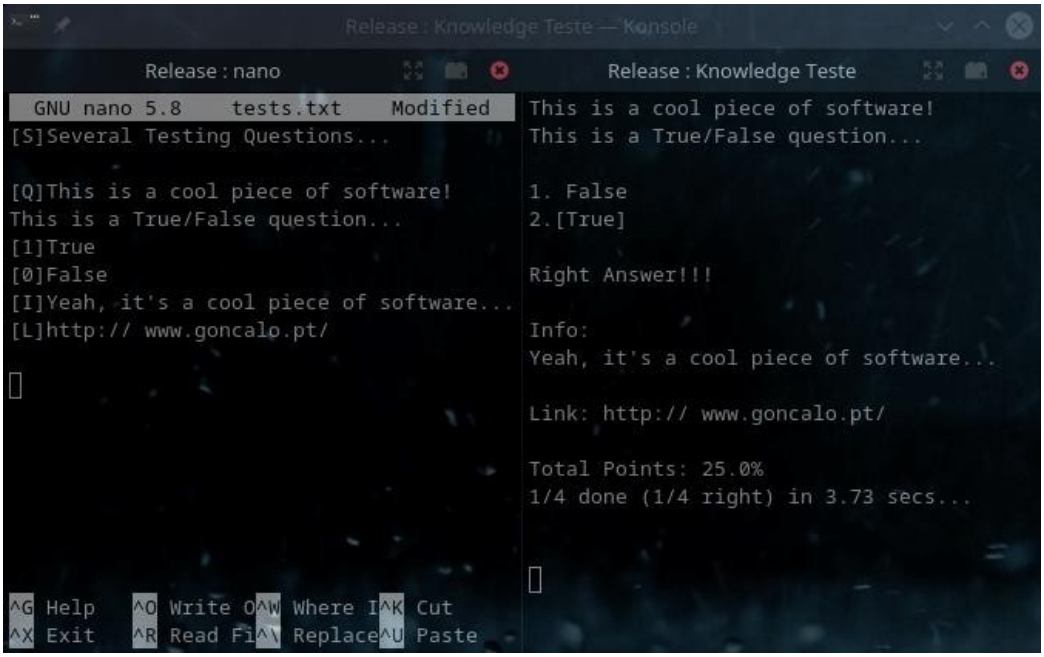
É de notar que responder com os números do teclado numérico também não é garantido, terá de ser com as teclas normais de números, as que estão por cima das letras no teclado.

Podemos ver abaixo como criar uma pergunta de escolha única:



Notem que a ordem inicial das questões não importa muito, porque por defeito a aplicação coloca as opções no ecrã de forma aleatória, mas o que podem ver acima é o suficiente para se criar um exame passível de ser executado.

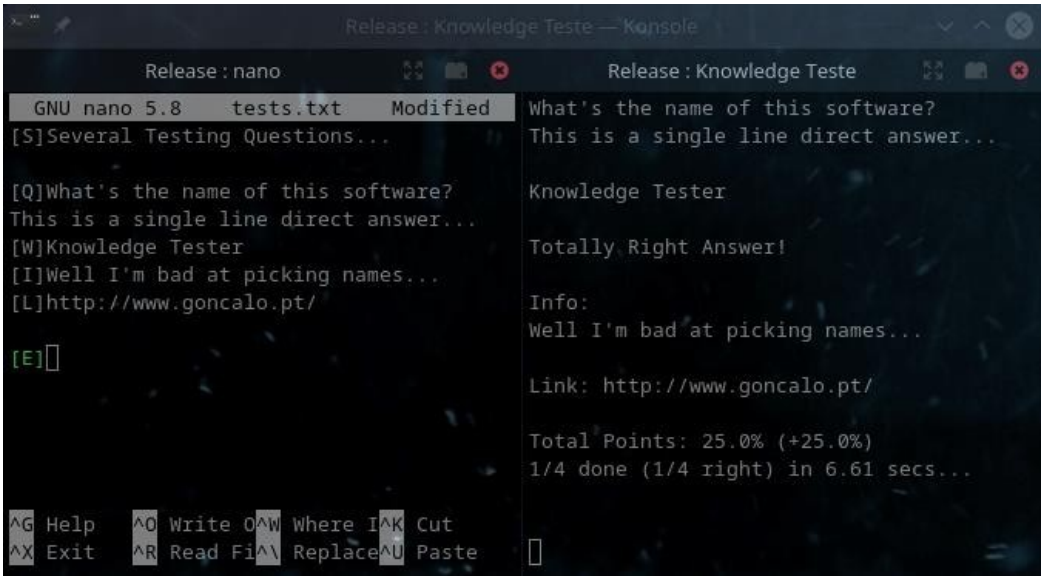
Podemos ver agora como criar uma pergunta de verdadeiro ou falso:



[W] – Direct Answer

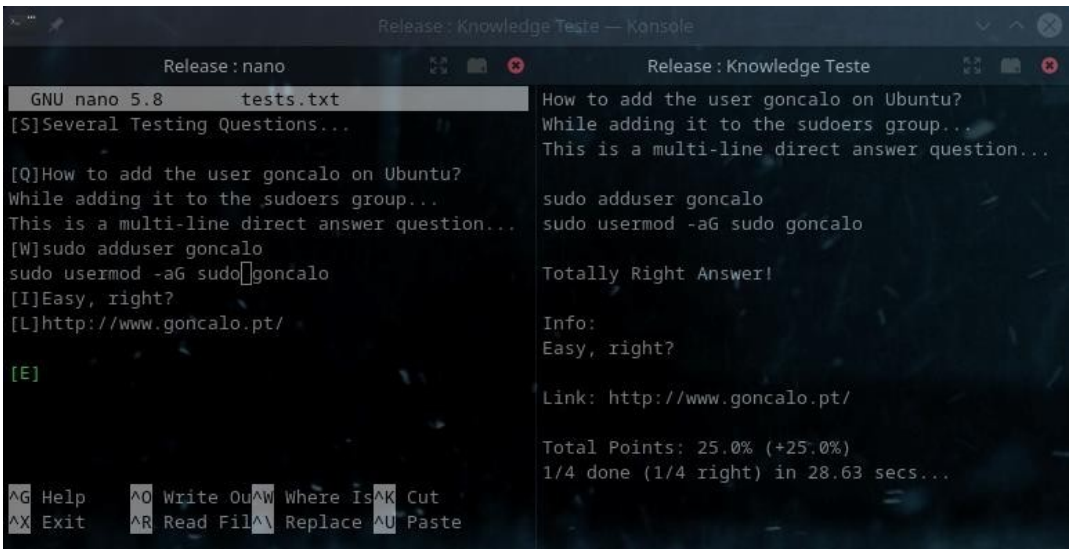
Quando uma ou mais linhas de texto estão precedidas por esta *tag*, será uma resposta de escrita, pelo que o utilizador tem a oportunidade de escrever frases inteiras de respostas antes de as submeter com a tecla "*Enter*", e a resposta só é finalmente submetida, quando submetemos uma linha completamente vazia, pelo que podemos submeter respostas de múltiplas linhas, quando estamos a por exemplo responder a laboratórios de múltiplas linhas de comandos;

Abaixo podemos ver como criar uma pergunta de resposta directa:



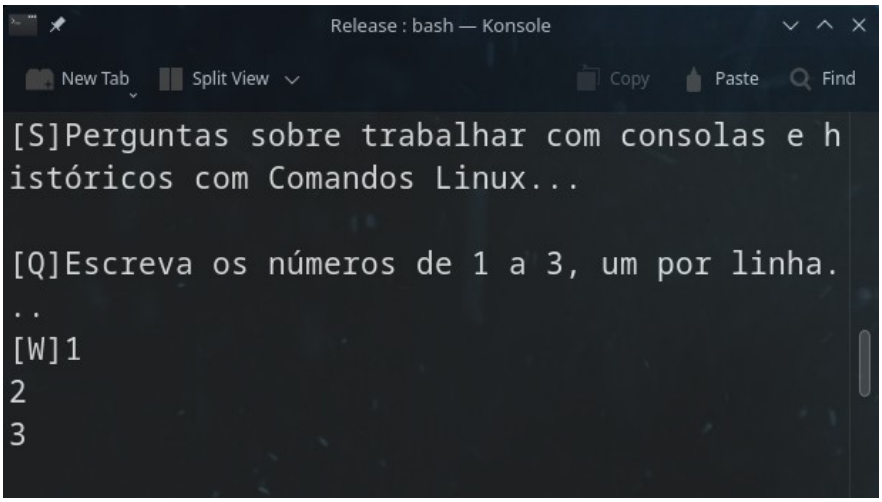
Podemos ver acima que basta colocar a resposta após uma *tag* "[W]" que ela é definida.

Abaixo podemos ver como definir uma pergunta de resposta directa multi-linha:



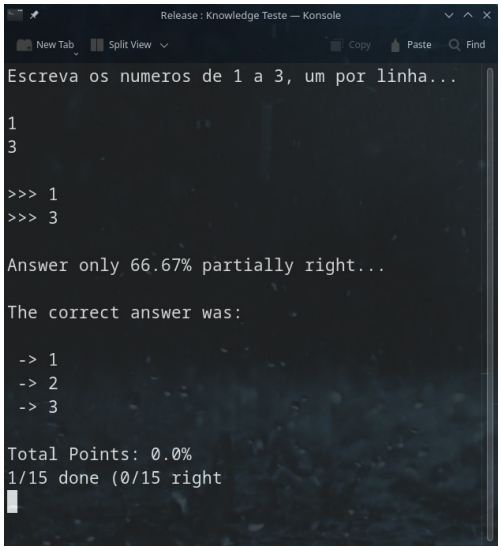
Vejamos agora, um exemplo, de uma resposta multi-linha mas directa.

Se quisessemos criar uma pergunta com apenas três linhas de resposta, faríamos assim:



O *software* perceberia de imediato ao ler o exame, de que a pergunta só estaria certa, se respondêssemos às três linhas completas.

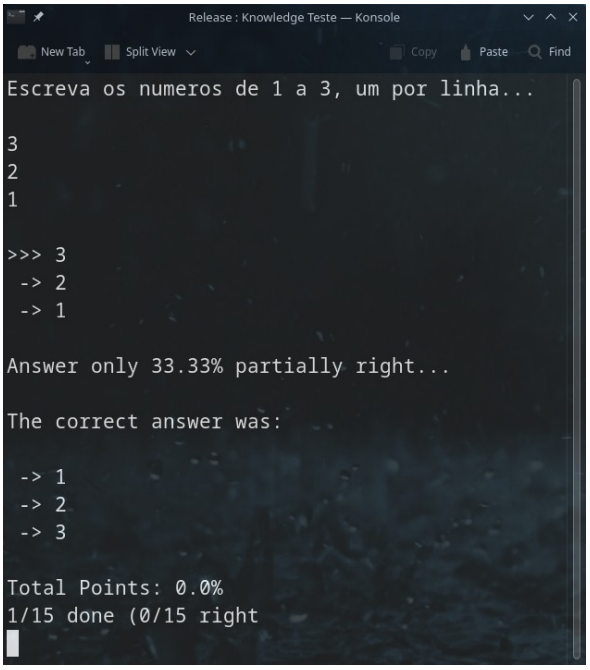
E neste caso, vamos simular uma resposta, mas respondendo só a duas:



Como podemos ver acima, ao respondermos só duas das três respostas certas, ele deu-nos 2/3 da nota (arredondados) correspondente a essa pergunta, e não aos 100% que a mesma nos daria.

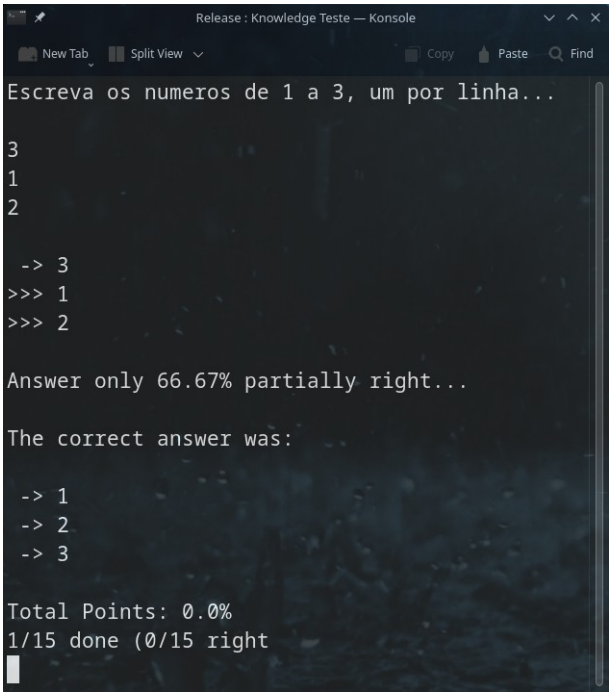
É de notar que as respostas teriam de ser dadas pela ordem certa, pois se fossem dadas com ordens erradas, o resultado seria diferente.

Abaixo temos as três respostas dadas de forma certa, mas na ordem errada:



Isto acontece pois só assim teria sentido, em especial no caso de estarmos a fazer laboratórios, em que podemos querer um seguimento de comandos, como num laboratório de consola de Linux, e os comandos com ordem errada, não estariam correctos, e nesse caso, o *software* avaliaria de forma automática qual a combinação de respostas que daria mais pontos, e marcá-la-ia como certa.

Assim, se respondêssemos 3, seguido de 1 e 2, ele marcaria certas as 1 e 2 juntas:



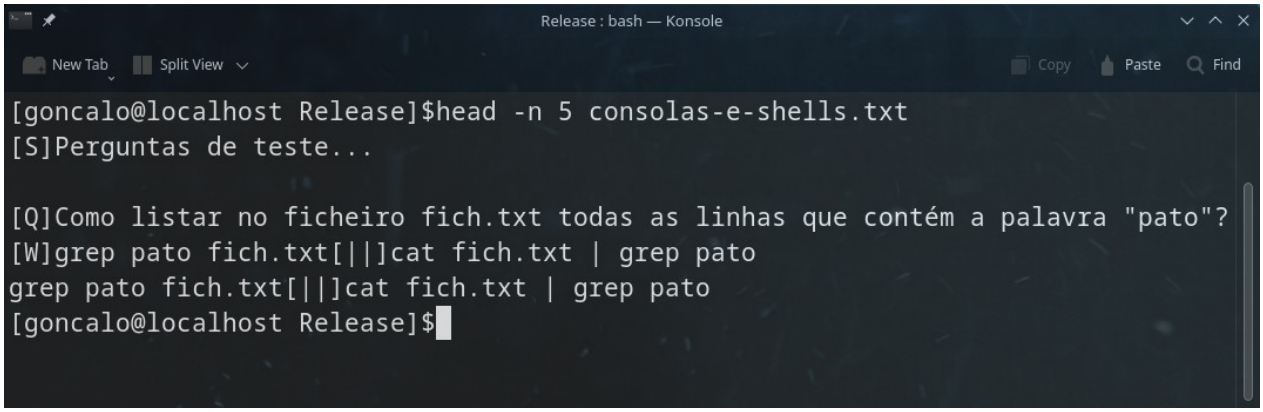
Relembro que isto é muito importante para fazer laboratórios, onde podemos memorizar um conjunto de comandos necessários para fazer algo, útil para quem treina para *Linux* ou até *Cisco IOS*, os *routers* mais famosos da actualidade, onde simulamos com este *software* exemplos práticos da vida real, provavelmente de horas de prática, em minutos, pois o *software* "força-nos" a repetir acções que levaríamos dias a encontrar novamente, várias vezes por hora, dependendo do número de vezes que usamos o *software* para praticar, claro está.

[|] - *Or* (Ou)

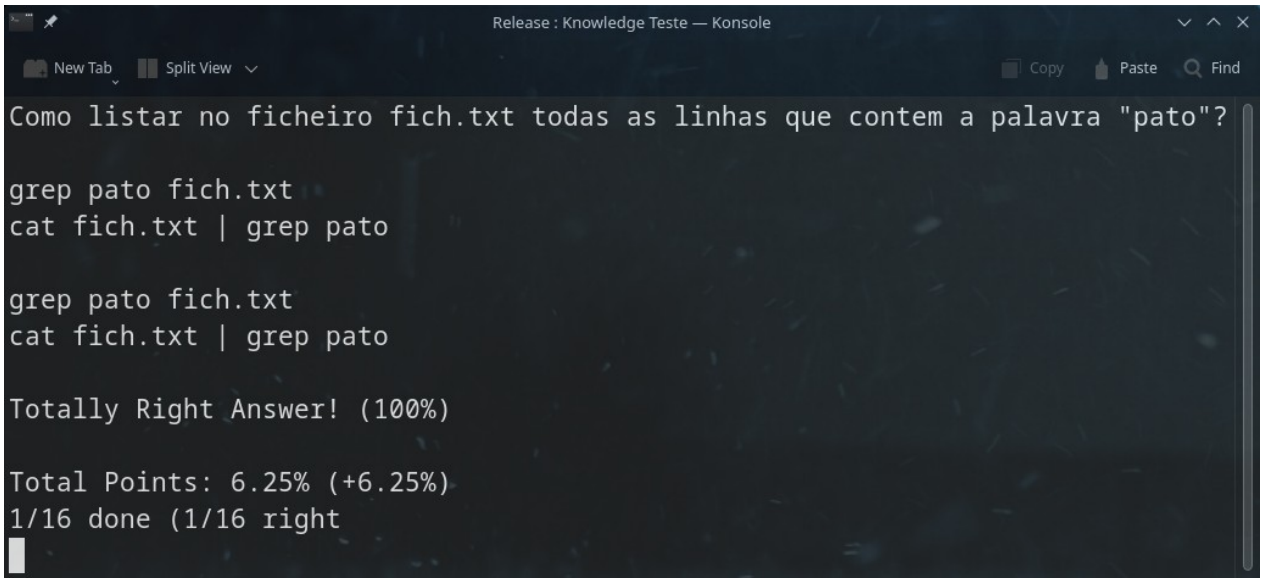
Esta tag é usada para definir múltiplas respostas possíveis por cada linha de resposta directa.

Ou seja, para os casos de pedirmos a um examinado que nos diga como executar um comando que tenha várias formas possíveis de responder certo, e queiramos que o *software* os avalie de forma automática para poupar trabalho ao examinador.

Vejamos um exemplo de resposta múltipla a ser definida:



E agora a mesma em funcionamento:



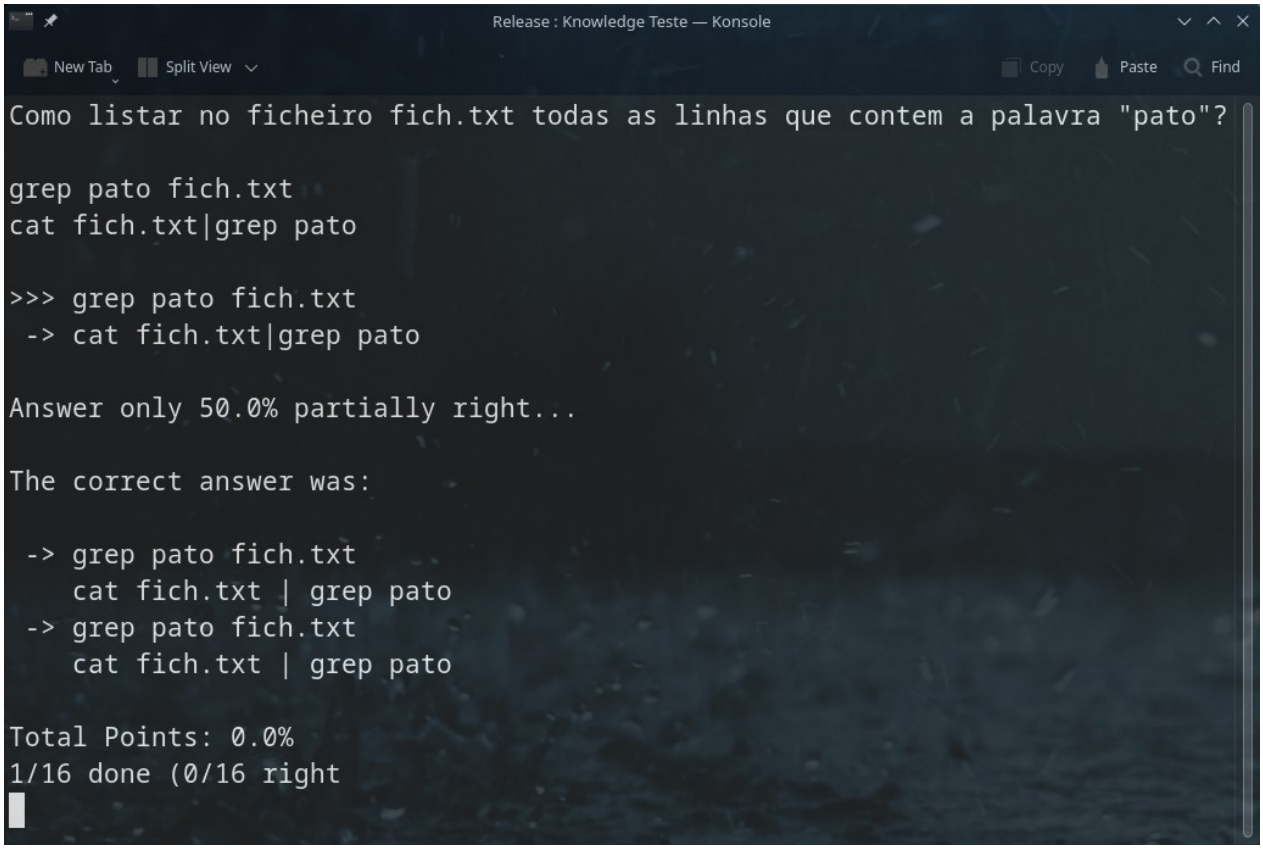
Podemos ver no exemplo anterior, que ambas as hipóteses eram certas.

É de notar que o *software* não detecta ainda se faltam espaços em certos comandos no meio, e talvez nem vá detectar, porque por exemplo, neste caso não haveria problema, mas se fizéssemos um "*ls *t>fich.txt*" não haveria problema em aceitar o ">" do meio com ou sem espaço antes, mas um "*ls *2>fich.txt*" já falharia, porque um número 1 ou 2 antes do ">" tem um sentido específico, e em *Linux* procurar ficheiros terminados em 2, teríamos de ter neste caso o "*ls *2 >fich.txt*" (com o espaço após o 2), que seria o mesmo que "*ls *2 1>fich.txt*".

Por isso, o *software* nunca poderia saber se deveria aceitar espaços ou não numas perguntas ou noutras, porque em certos casos sim, noutros não, e um *software* não é suposto saber isso.

Podemos na mesma criar duas respostas possíveis, uma com o espaço antes do sinal “>” e outra sem, para aceitar ambas. Mas será mais fácil indicar aos examinados para colocar sempre um espaço antes ou depois do tal sinal.

Se falharmos uma das perguntas, ele mostra-nos quais as opções que teriam sido aceites como válidas em cada linha, sendo que cada linha começa na explicação por “->”:



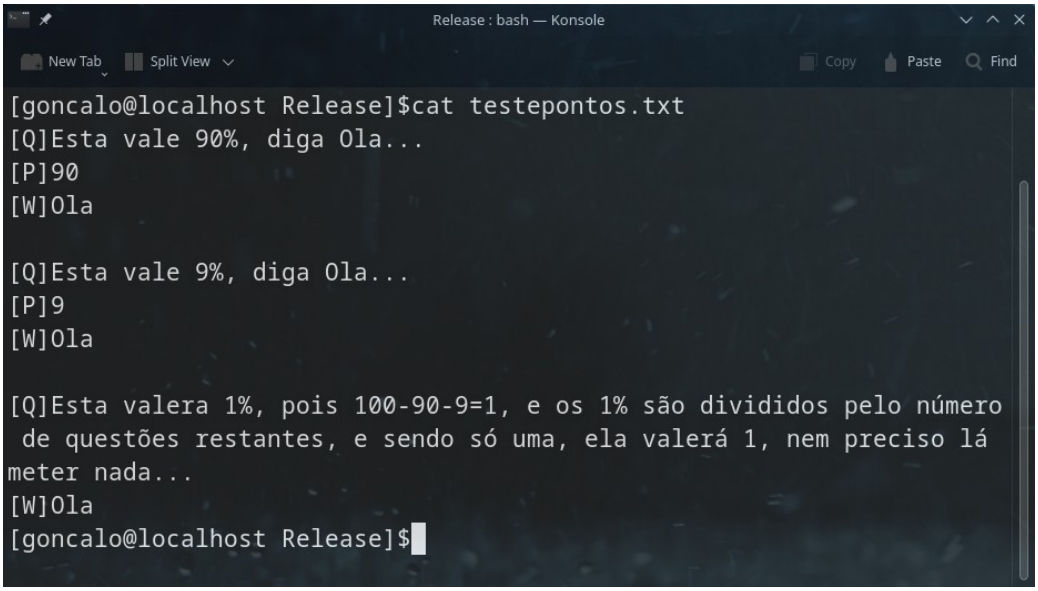
É de notar que ao definirmos múltiplas possibilidades de resposta por linha, temos de ter todas as possibilidades de resposta de cada linha, sem espaços, porque se temos um “*ola[||]hello[||]*”, e após uma mudança de linha, um “*hallo*”, ele assumiria o “*hallo*” como uma resposta para a segunda linha. Devem estar todas as alternativas de cada linha, numa única linha, para o *software* saber onde acaba cada linha.

[P] – Pontuação (só no modo Gráfico)

Esta *tag* permite-nos definir a pontuação de cada pergunta, pois poderemos ter uma pergunta que falha 10% do total, por ser multi-linha com 10 linhas, e 90 outras perguntas que falham 1% cada por terem só uma linha de resposta (ou qualquer outro critério).

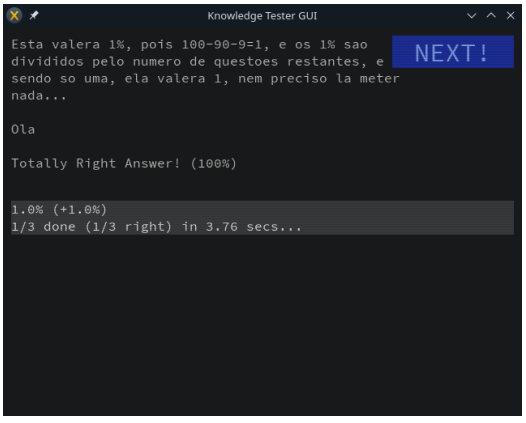
É importante esclarecer que se nunca for usada esta *tag*, a pontuação será sempre definida de maneira simples: dividem-se os 100% finais pelo número de questões, sendo que se forem 10 questões, valerá por defeito 10% cada uma, se forem 100, 1% cada uma, e se forem 1000, 0,1% cada uma.

Se quisermos alterar o valor de uma questão (não precisamos de alterar em todas), fazemos isso da seguinte maneira:

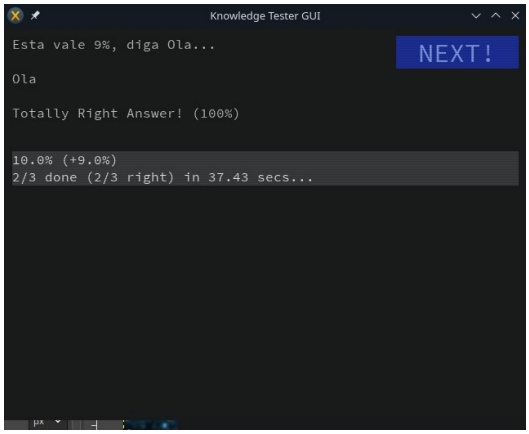


No exemplo acima, definimos a primeira questão como valendo 90% do exame, e neste caso será 90% do exame para uma única linha (teria mais sentido se fosse uma questão com 9 linhas de resposta e 10% por cada uma), a segunda questão valendo 9%, e a última sem valor definido, pelo que acabará por receber a totalidade do restante, que é de 1%, que será dividido pelo total de perguntas restantes, ou seja uma, dando 1%.

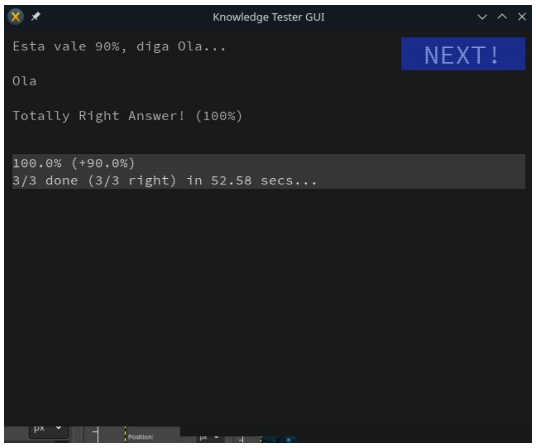
Vejamos em funcionamento, no modo gráfico, pois só funciona em modo gráfico, por falta de tempo meu:



Calhou aparecer em primeiro lugar a restante com os 1% que restavam. Vejamos a segunda:



Como podemos ver, a segunda era a dos 9%, e deu 9% de avaliação. Vejamos a terceira:



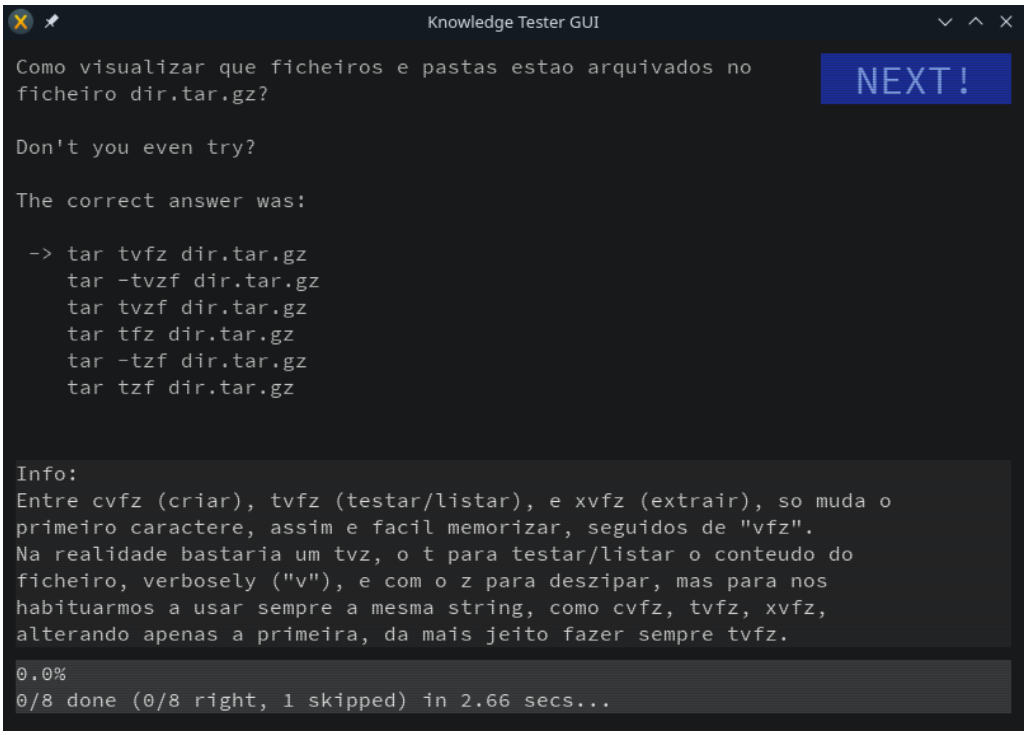
Por fim, na terceira que nos apareceu, a pontuação era de 90% caso estivesse certa, pois tinha sido a que definimos como tendo o valor de 90 na *tag* "*[P]*".

É importante relembrar que **esta funcionalidade das pontuações customizadas só funciona no modo gráfico**, pois eu deixei o projecto a meio até 2025, porque tenho muitos projectos pessoais, os quais gosto mais do que este (que é uma mera ferramenta de trabalho para mim), e aos quais quero dedicar mais tempo neste ano e meio que vem.

Por isso, evitem tentar usar no modo de terminal.

[I] – Info

Aqui definimos uma pequena ajuda, ou seja, quando o utilizador responde, verá após a sua resposta um texto com esta ajuda (quando não estamos no modo de exame claro está), que nos ajudará a aprender mais sobre a resposta, porque o utilizador deve ver alguma explicação sobre a resposta para compreender e não simplesmente memorizar:



Acima temos um exemplo de como se pode aprender após cada resposta.

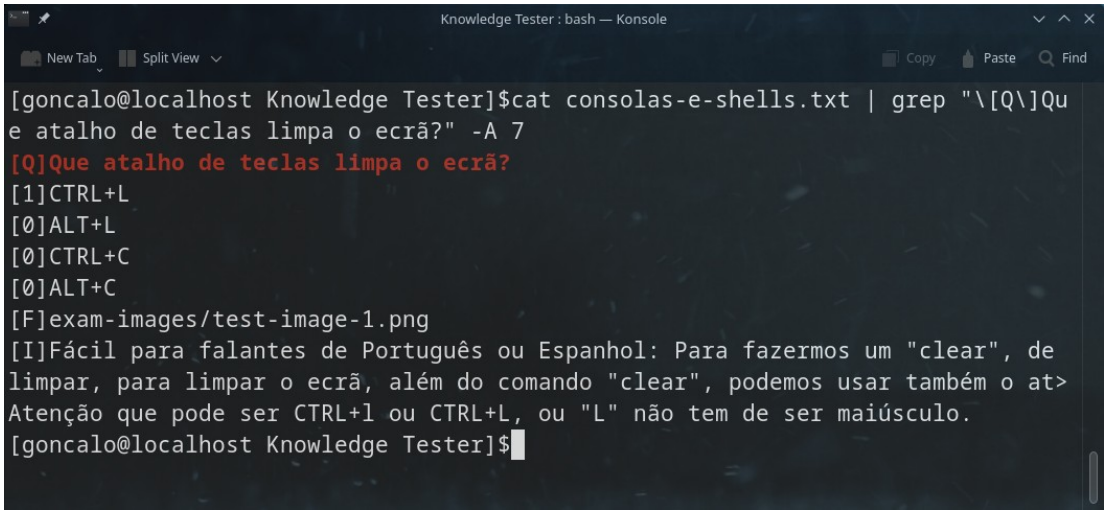
[F] – Image File

Esta é uma funcionalidade que nos permite adicionar imagens a um exame, para os casos de termos perguntas que requerem que haja alguma interpretação de algum tipo de imagem ou diagrama, como por exemplo em exames de prática para cursos de redes como o *CCNA*, por exemplo.

Nesses casos é importante termos imagens nos exames.

Não tive ainda tempo de criar um sistema de embeber imagens dentro do próprio ficheiro do exame, pelo que de momento nesta versão, ainda só funciona com imagens em separado, que terão de ser distribuídas com o próprio exame em si.

Na criação dos exames, é muito fácil invocar o aparecimento dessa imagem numa pergunta:



Como podemos ver acima, basta-nos colocar a *tag* "*[F]*" algures na pergunta, para que o *software*, quando chega a essa pergunta, colocar a imagem visível junto com a mesma.

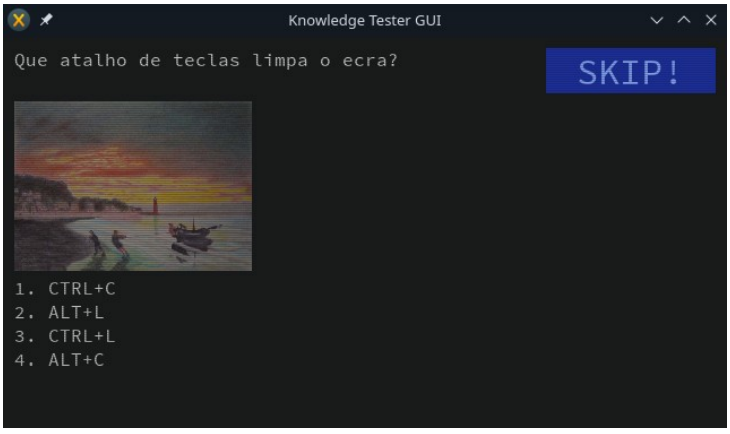
Repito que neste caso, teríamos de distribuir o exame junto com a pasta "*exam-images*" (apenas por ter sido o nome dado no exemplo acima), com a imagem "*test-image-1.png*" lá dentro, para que o exame ao ser executado a pudesse ir buscar.

Isto é facilmente feito, basta distribuirmos o exame junto com a pasta das imagens, tudo junto num ficheiro comprimido/zipado.

Mais tarde, lá para 2025, criarei um sistema que ao gerar o exame, embeberá as imagens todas dentro do ficheiro do exame, tudo encriptado, para que possa ser distribuído junto com as mesmas num só ficheiro.

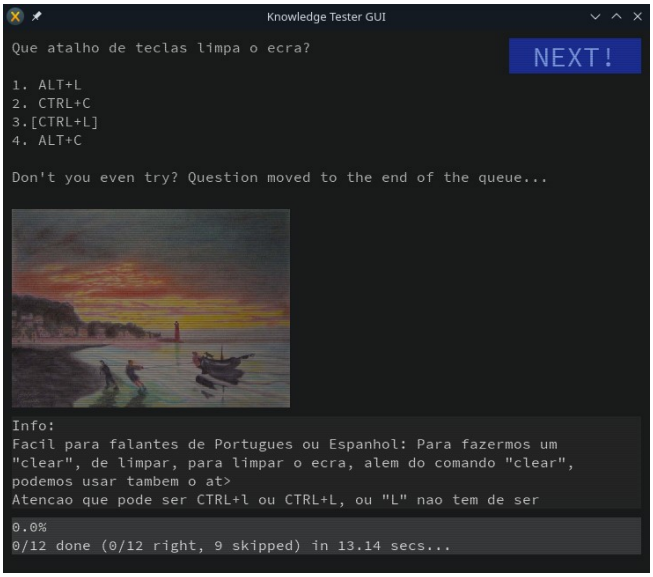
Não é difícil para mim (já o faço em coisas como no *Game Engine* em que até são embebidas no próprio executável), mas falta-me tempo de momento para o fazer em 2023.

Durante a execução, seria isto que veríamos, ao chegar a essa pergunta:



Como podemos ver acima, logo a seguir à pergunta, aparece a imagem, e ela vê o seu tamanho ser ajustado em tempo real consoante o tamanho da janela do *software*.

Mesmo no *feedback* da questão, continua a aparecer a imagem, para podermos verificar por exemplo um diagrama e perceber porque errámos a questão (nos exemplos que apresentei está apenas uma pintura minha impressionista mas serve de exemplo):



É assim muito fácil inserir imagens em exames nossos, para certos tipos de exercícios.

Mas de momento como eu não uso e amigos meus que usam o *software* não têm precisado de usar imagens, e eu desenvolvo o *software* em especial para uso próprio e numa *need to use basis*, não preciso de desenvolver mais do que o que já está de momento.

[L] – Link

Após esta *tag*, Temos um endereço para um *website* ou página na *Internet* com mais dados sobre o assunto, isto já a pensar nalguma versão com interface gráfica que vá ser desenvolvida um dia, mas também relembrando que em várias aplicações de terminal, como a *Konsole* do *KDE*, podemos clicar no *link* mesmo na *shell* com o rato, e o *link* abrirá no *browser*.

Esta é uma funcionalidade que nunca desenvolvi, imaginada em 2019 quando fiz o rascunho deste futuro projecto, e que supostamente um dia fará com que apareça um endereço de um *website* (um *link*), com a ajuda, para quem quiser possa clicar e visitar o site e aprender mais.

Talvez lá para 2025 adicione esta *feature*.

[E] – End

Esta *tag* define o fim do ficheiro de perguntas, pelo que tudo o que vier após esta *tag* será ignorado pelo *software*, apesar de esta *tag* ser opcional e os exames poderem ser executados sem ela estar presente no exame em si.

Conclusão sobre a criação de exames

Sendo assim, basta criar um ficheiro de perguntas, obedecendo a estas regras, e depois correr os exames, e tudo funcionará.

Não há muito mais simples que isto, que o poder criar exames num simples editor de texto, para ser usado por qualquer pessoa depois.

Os tipos de perguntas

De momento existem apenas dois tipos de questões principais, as de escolha múltipla e resposta directa, e que se dividem no fundo em quatro:

Escolha Múltipla

Nesta são apresentadas várias opções ao utilizador, estando apenas uma certa, e tendo de escolher a opção certa.

Verdadeiro ou Falso

Esta é uma variante da de escolha múltipla, e no fundo consiste numa pergunta, e temos como resposta apenas “Verdadeiro” ou “Falso”, sendo que uma tem “[1]” antes e a outra “[0]” quando é criada. Desta forma, estamos a ter uma pergunta de Verdadeiro ou Falso, com escolha múltipla.

Resposta Directa

Aqui é-nos dada a oportunidade de responder por escrito a uma pergunta, e ser-nos-á depois confirmado se a resposta está certa ou errada.

Laboratório (Resposta Directa Multi-Linha)

Esta é uma variante da resposta directa, mas com uma resposta multi-linha, onde o sistema depois verifica quais as linhas em que acertámos da resposta original, para nos gerar uma pontuação.

Serão adicionadas novas funcionalidades mais tarde, mas estas são as mais importantes.

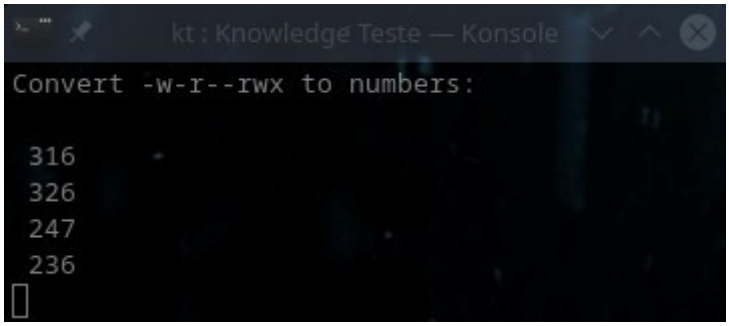
Temos agora um resumo abaixo de cada uma em separado.

Explicação sobre os vários tipos de perguntas

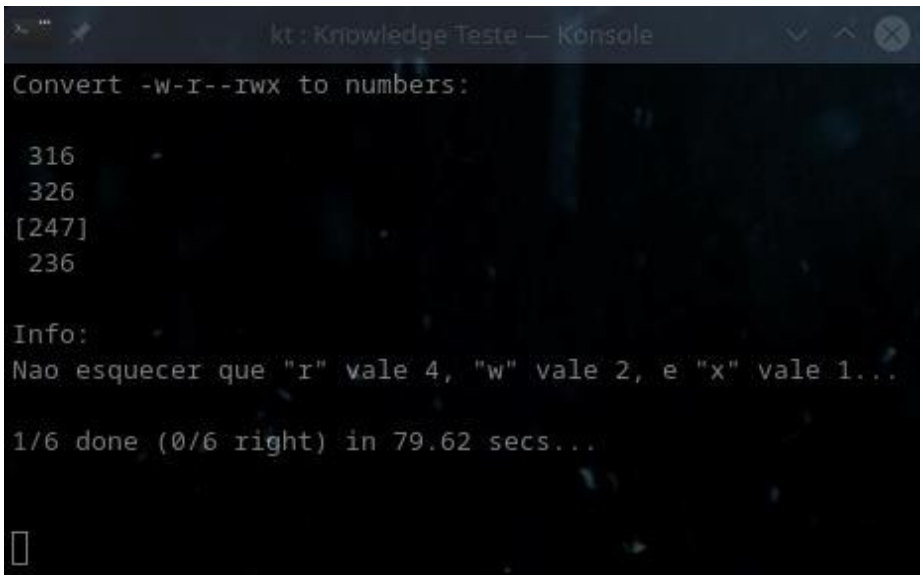
Escolha Múltipla

Nesta variante de perguntas, são apresentadas várias opções ao utilizador, estando apenas uma certa, e tendo de escolher a opção certa.

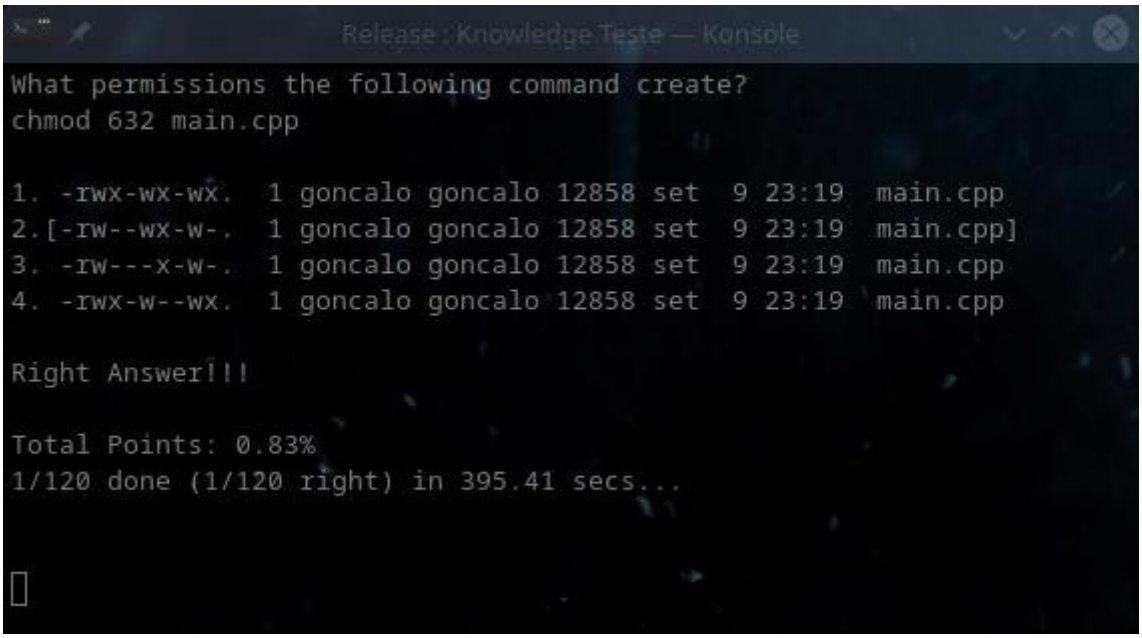
Abaixo temos como exemplo uma pergunta no modo "*--mentalMode*", em que não aparecem as opções numéricas para clicarmos para escolher a questão, onde nos basta apenas pressionar uma tecla para visualizar se acertámos no resultado ou não:



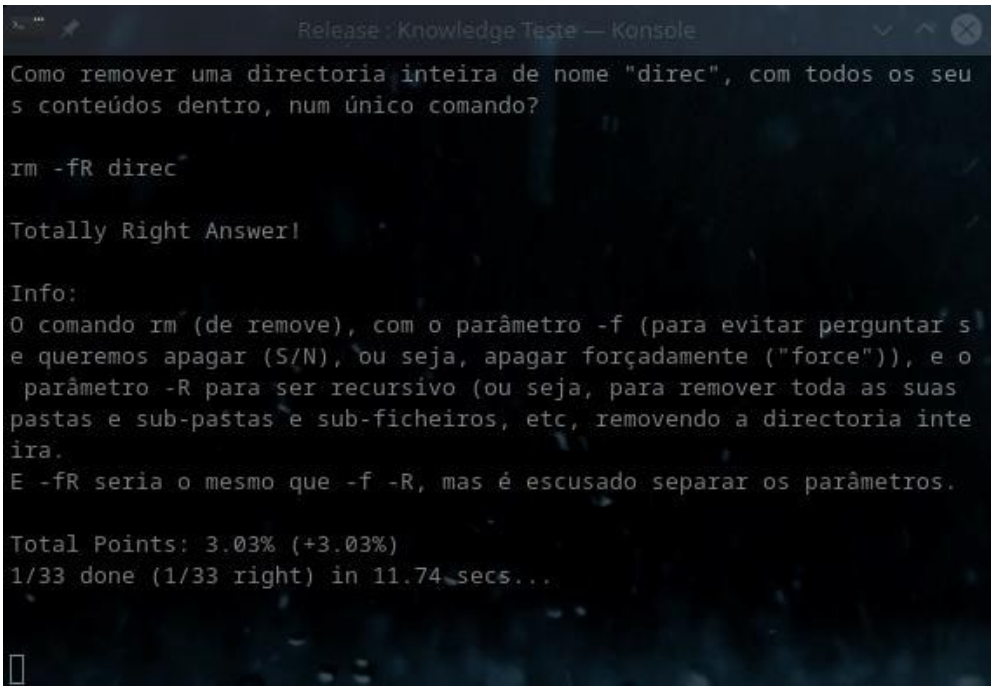
Ao clicarmos numa tecla, a opção certa ganha realce e ficamos a saber se acertámos ou não, mentalmente, pois o *software* não saberá pois não tem *feedback* por parte do utilizador, pois ele se limita a clicar numa tecla para prosseguir:



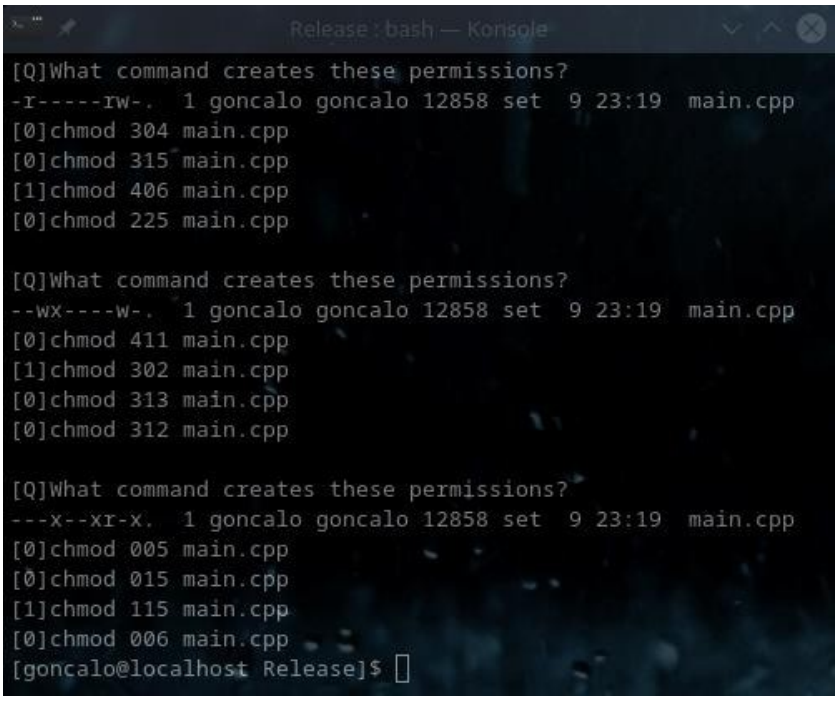
No modo de exame ou no modo normal, já aparecem as opções para escolha e as respostas já são avaliadas:



E quando respondemos a alguma pergunta que tenha ajuda, aparece-nos essa ajuda sobre a questão, além da informação sobre se está certa ou errada:



Estas perguntas de escolha múltipla são criadas de forma fácil, com uma *tag* "[0]" a representar as respostas erradas, e uma "[1]" a representar a resposta certa, e o programa trata depois de baralhar as opções:



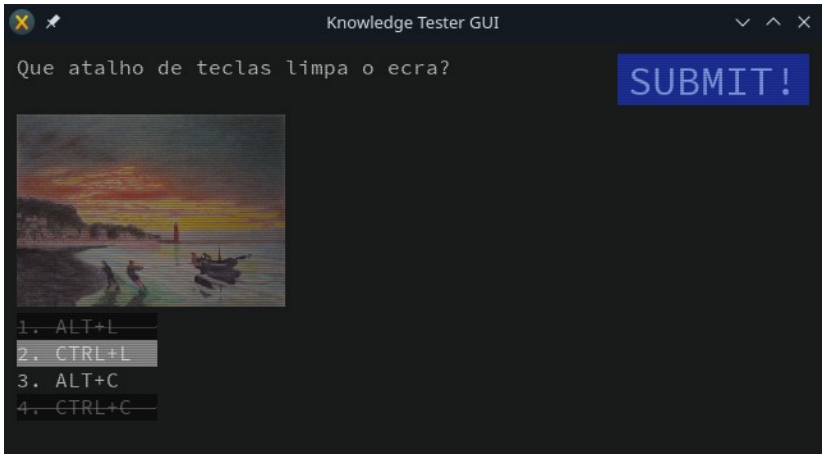
As respostas acima são geradas dinamicamente com a funcionalidade do parâmetro "*--createFilePermissions*".

No caso de colocarmos múltiplas respostas numa só linha, é escolhida uma delas de forma aleatória, para ser incluída como opção na resposta, pois só uma poderá aparecer certa, por isso se uma pergunta tiver várias respostas possíveis na mesma linha, usando a *tag* "*[||]*", só uma delas aparecerá, e será escolhida de forma aleatória pelo *software*.

As perguntas de escolha múltipla na interface gráfica

Na interface gráfica, o sistema é muito semelhante, mas temos funcionalidades extras com o uso do rato.

Vejamos a opção abaixo:



Como podemos ver acima, podemos marcar as opções erradas com o botão direito do rato, ficando as mesmas sob uma sombra e com um risco a meio. Isto ajuda-nos a eliminar as que estão erradas no meio de muitas opções de forma fácil, e ajuda-nos a achar a certa por exclusão de partes.

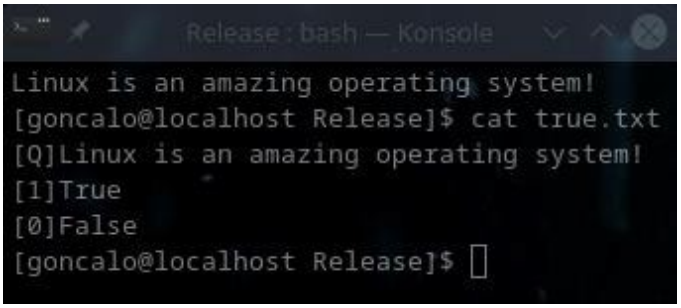
Em simultâneo, com o botão esquerdo do rato, marcamos a opção (acima a "*CTRL+L*") a branco, como certa, e será essa que o *software* avaliará quando submetermos o fomulário.

Podemos submeter o formulário no botão "*Submit*" ou com um simples pressionar da tecla *Enter*, ou com a combinação "*CTRL+Enter*".

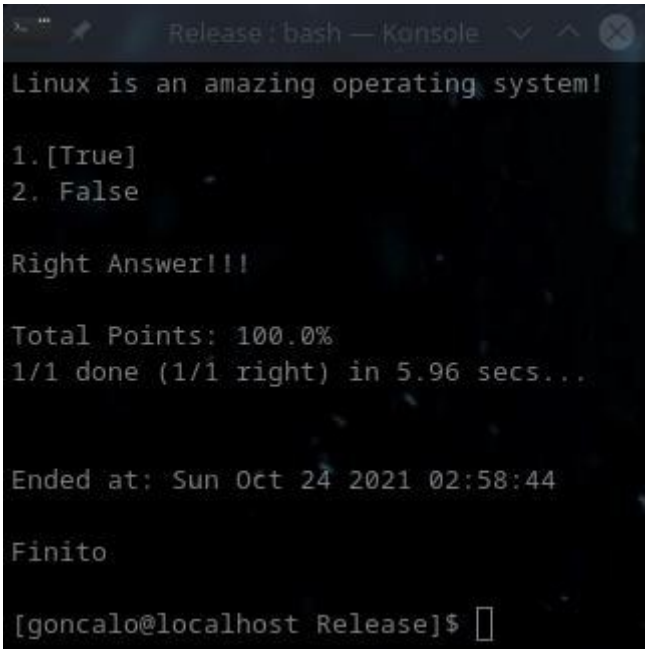
Verdadeiro/Falso

Esta é uma variante da de escolha múltipla, e no fundo consiste numa pergunta, e temos como resposta apenas "Verdadeiro" ou "Falso", sendo que uma tem "[1]" antes e a outra "[0]" quando é criada.

Desta forma, estamos a ter uma pergunta de Verdadeiro ou Falso, com escolha múltipla:



A configuração acima, depois dá-nos um exame com perguntas simplificadas de “Verdadeiro” ou “Falso”, onde no fundo são apenas duas respostas num teste “de cruzinhas”.



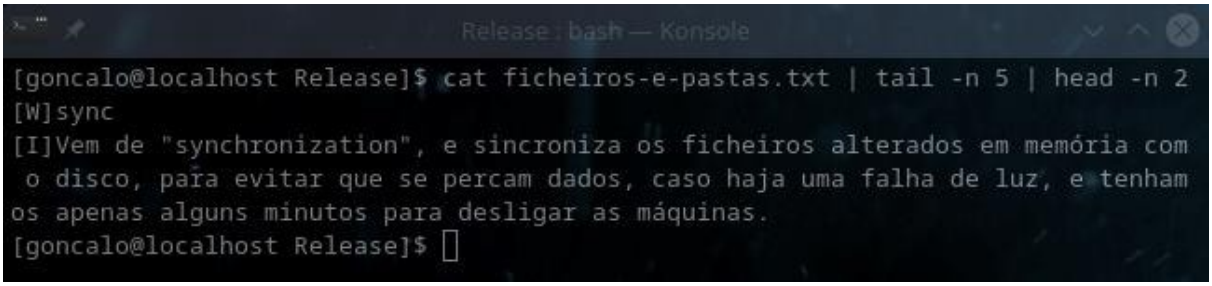
É tão fácil quanto isto…

Resposta directa

Esta é uma funcionalidade que testa melhor os nossos conhecimentos, pois obriga-nos a escrever a resposta por inteiro, e é útil na memorização, por exemplo, de comandos de *Linux*.

Aqui é-nos dada a oportunidade de responder por escrito a uma pergunta, e ser-nos-á depois confirmado se a resposta está certa ou errada.

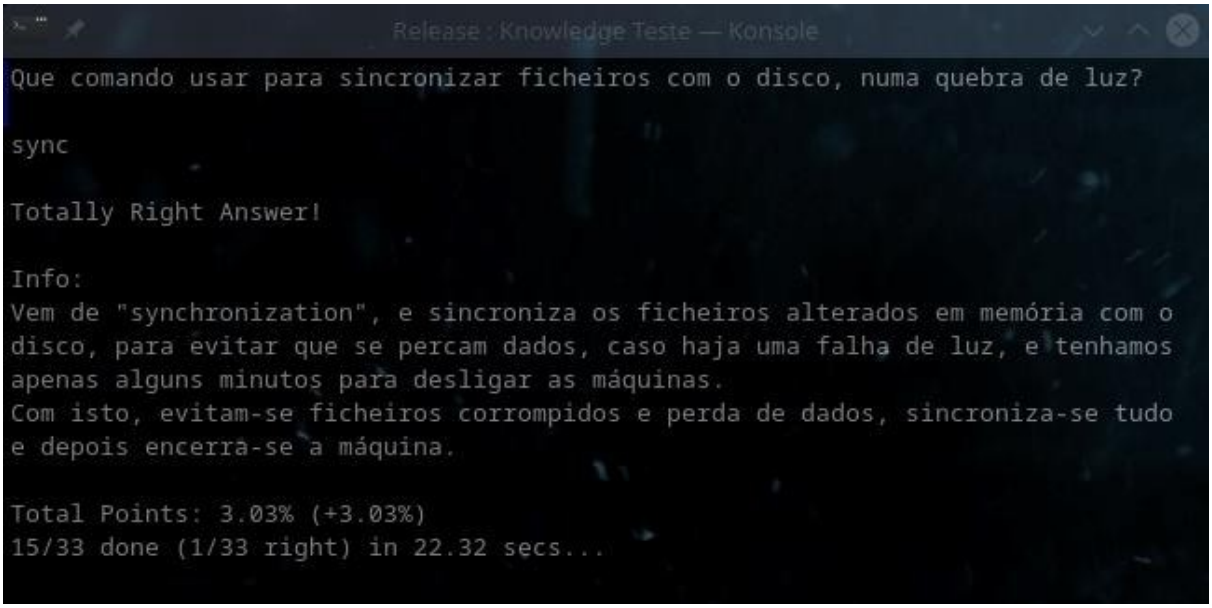
Ela é criada substituindo a *tag* certa de “[1]” por “[W]”, de “*Written*”:



Ela dá-nos depois também a informação sobre se está a resposta correcta ou não, bem como a informação sobre a mesma, criada com a tag “[I]” aquando da criação do exame.

Este tipo de perguntas é fácil de criar, basta que no editor coloquemos uma *tag* “[W]” (de “*Written answer*”) ao invés das típicas [1], para a resposta certa, e basta essa única *tag*.

E a resposta será dada directamente, por escrito:



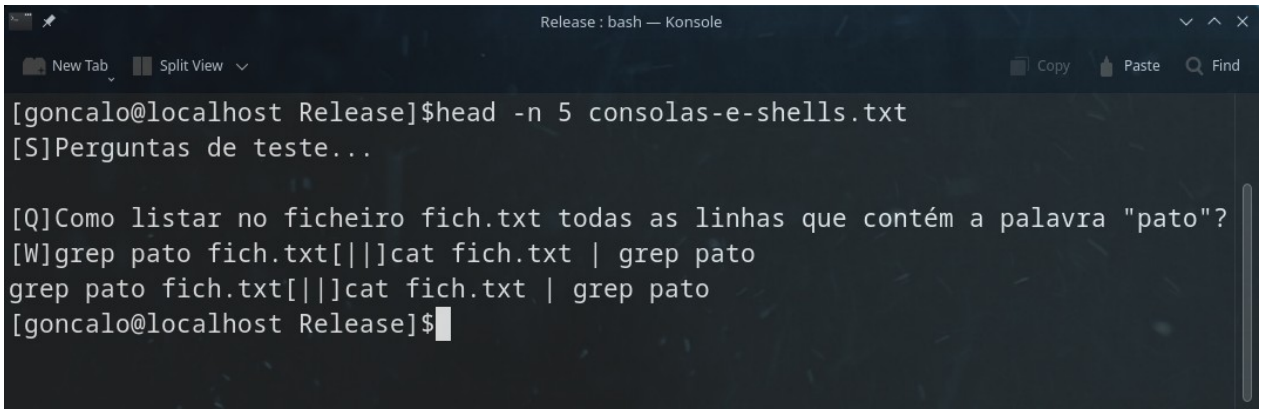
O *software* está preparado para identificar respostas mesmo que tenham um número diferente de espaços entre palavras, mas se um caractere que não espaços estiver errado, a resposta será marcada como errada e não contabilizada na pontuação, e caberá ao examinador depois rever o relatório gerado no fim do exame para corrigir a pontuação, pelo que por norma, a pontuação do aluno será sempre superior após revisão do examinador, que terá de analisar também erros de escrita, não perceptíveis pelo *software*.

O uso de múltiplas respostas possíveis com a *tag [|]* (*Or* - Ou)

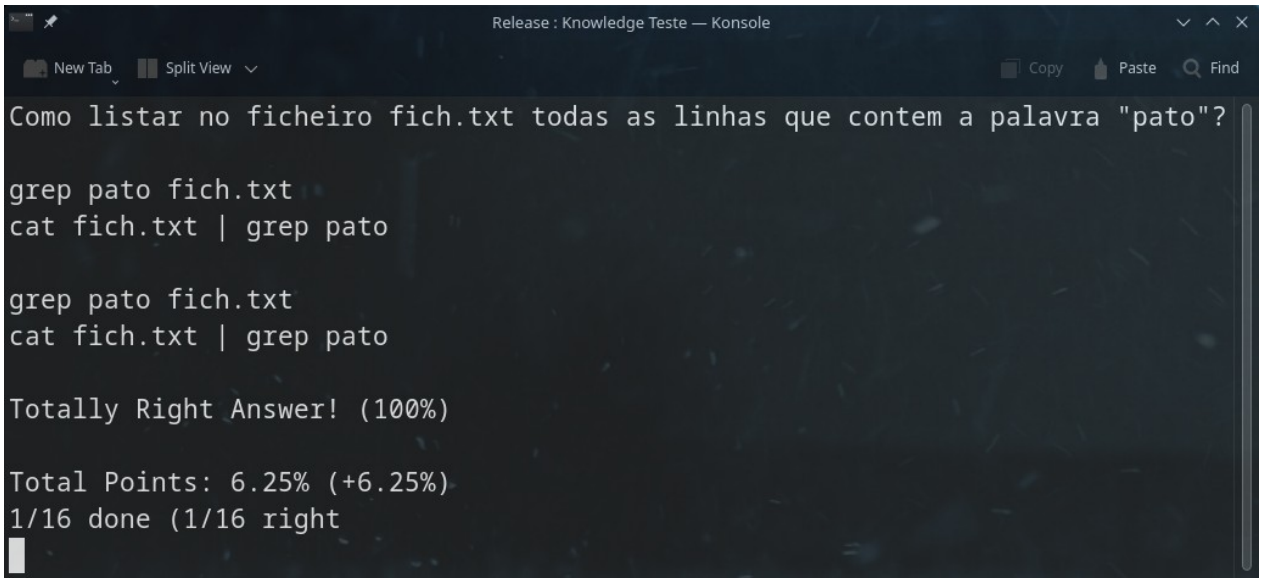
Relembro que esta tag é usada para definir múltiplas respostas possíveis por cada linha de resposta directa.

Ou seja, para os casos de pedirmos a um examinado que nos diga como executar um comando que tenha várias formas possíveis de responder certo, e queiramos que o *software* os avalie de forma automática para poupar trabalho ao examinador.

Vejamos um exemplo de resposta múltipla a ser definida:



E agora a mesma em funcionamento:



Podemos ver no exemplo anterior, que ambas as hipóteses eram certas.

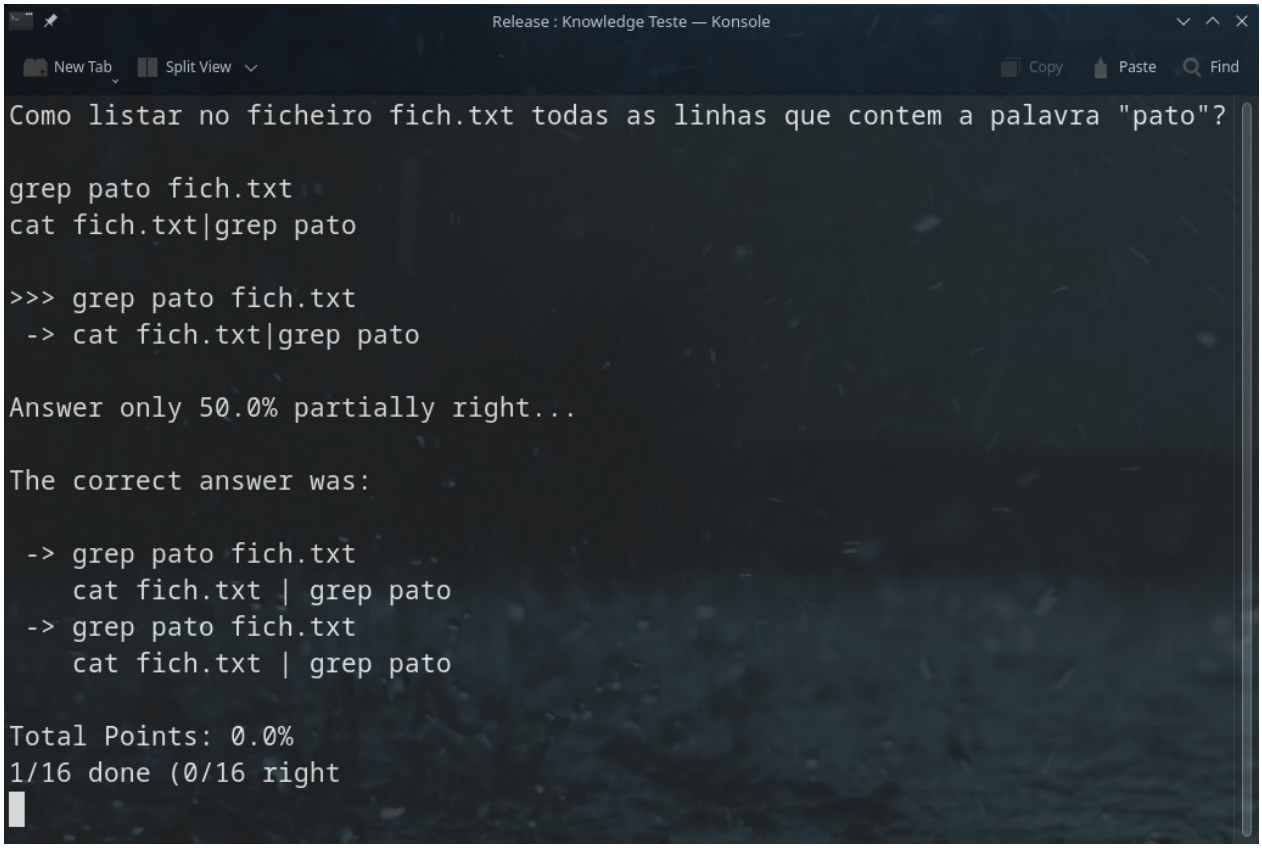
É de notar que o *software* não detecta ainda se faltam espaços em certos comandos no meio, e talvez nem vá detectar, porque por exemplo, neste caso não haveria problema, mas se fizéssemos um "*ls *t>fich.txt*" não haveria problema em aceitar o ">" do meio com ou sem espaço antes, mas um "*ls *2>fich.txt*" já falharia, porque um número 1 ou 2 antes

do “>” tem um sentido específico, e em *Linux* procurar ficheiros terminados em 2, teríamos de ter neste caso o “*ls *2 >fich.txt*” (com o espaço após o 2), que seria o mesmo que “*ls *2 1>fich.txt*”.

Por isso, o *software* nunca poderia saber se deveria aceitar espaços ou não numas perguntas ou noutras, porque em certos casos sim, noutros não, e um *software* não é suposto saber isso.

Podemos na mesma criar duas respostas possíveis, uma com o espaço antes do sinal “>” e outra sem, para aceitar ambas. Mas será mais fácil indicar aos examinados para colocar sempre um espaço antes ou depois do tal sinal.

Se falharmos uma das perguntas, ele mostra-nos quais as opções que teriam sido aceites como válidas em cada linha, sendo que cada linha começa na explicação por “->”:



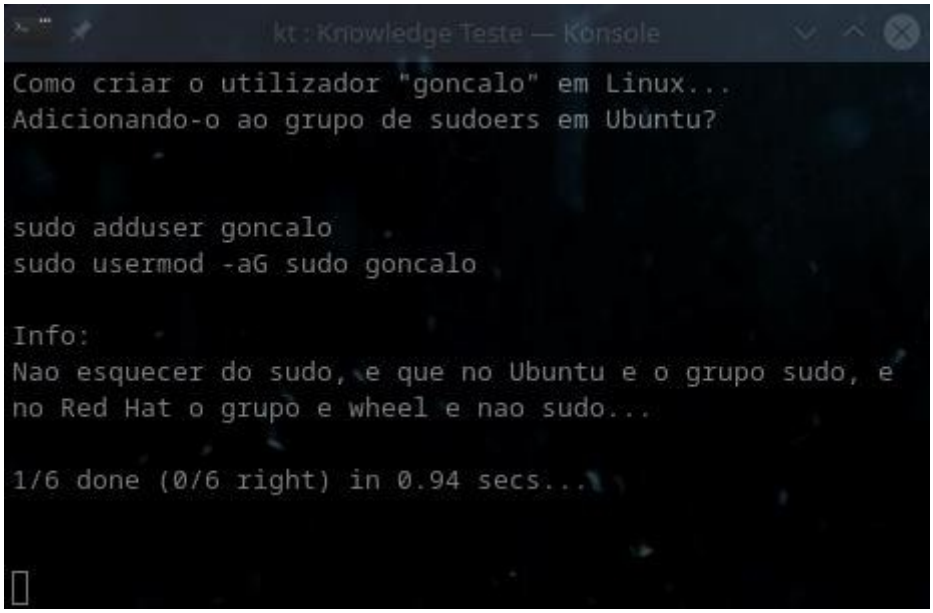
É de notar que ao definirmos múltiplas possibilidades de resposta por linha, temos de ter todas as possibilidades de resposta de cada linha, sem espaços, porque se temos um “*ola[||]hello[||]*”, e após uma mudança de linha, um “*hallo*”, ele assumiria o “*hallo*” como uma resposta para a segunda linha. Devem estar todas as alternativas de cada linha, numa única linha, para o *software* saber onde acaba cada linha.

Laboratório (Resposta Directa Multi-Linha)

Esta é uma variante da resposta directa, mas com uma resposta multi-linha, onde o sistema depois verifica quais as linhas em que acertámos da resposta original, para nos gerar uma pontuação.

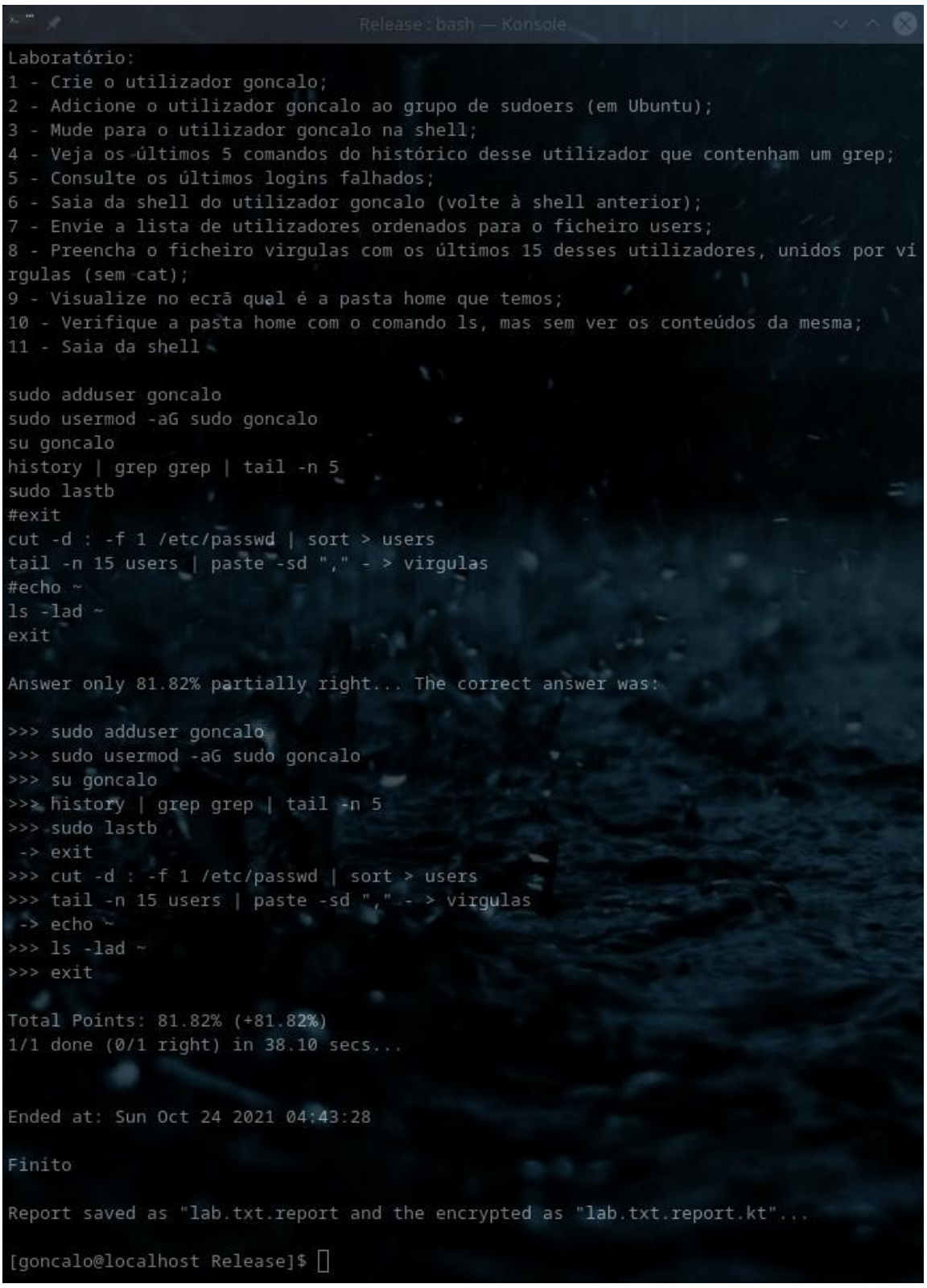
Tal como nas respostas de uma única linha, podemos usar aqui também a *tag* "*[||]*" (a *Or*, o operador lógico matemático "*V*"), para permitir múltiplas opções de resposta por cada linha.

Na imagem abaixo vemos uma resposta a uma pergunta multi-linha, no *mental mode*, para memorização:



Este modo é especialmente útil no treino para certificações com laboratórios, em que teremos de saber de cor processos inteiros com múltiplas linhas de comandos, como certificações *Red Hat*, ou até *Cisco*, entre outras, ou se quisermos pura e simplesmente não nos esquecer de como fazer certas operações que raramente executamos mas que devemos ter memorizadas.

Quando no modo normal ou de exame, já nos é pontuada a resposta multi-linha, onde nos é indicado quais das linhas da resposta estavam certas, e a pontuação da pergunta será dividida entre o número de linhas e a pontuação final contabilizada:



Se virem bem na respostas acima, quando as linhas diferem, aparecem com apenas “->” antes da resposta que deveria ser certa, enquanto que as que estão iguais às respostas certas, aparecem com “>>>” antes.

Pontuações

Também é dividida a pontuação total da resposta pelo número de linhas, para se calcular a pontuação final de acordo com o número de linhas que estavam correctas, sendo que neste caso como só existia uma pergunta (era um laboratório de exercício único), ela valeria 100%, e assim, a pontuação final ficou como 81,82% (9 em 11 linhas certas).

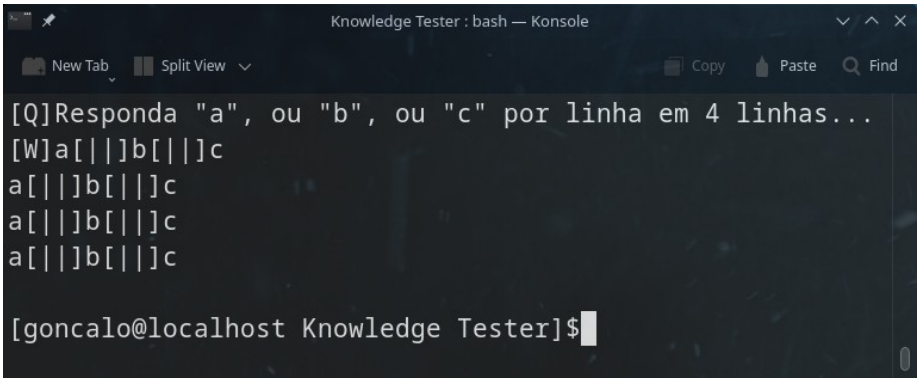
Pontuação customizada de perguntas – a *tag “[P]”*

Só para relembrar que se consultarem a secção referente à *tag* “*[P]*”, poderão ver como dar pontuações customizadas a cada pergunta, que é possível e não têm de ter todas as mesma pontuação.

Múltiplas opções válidas por linha

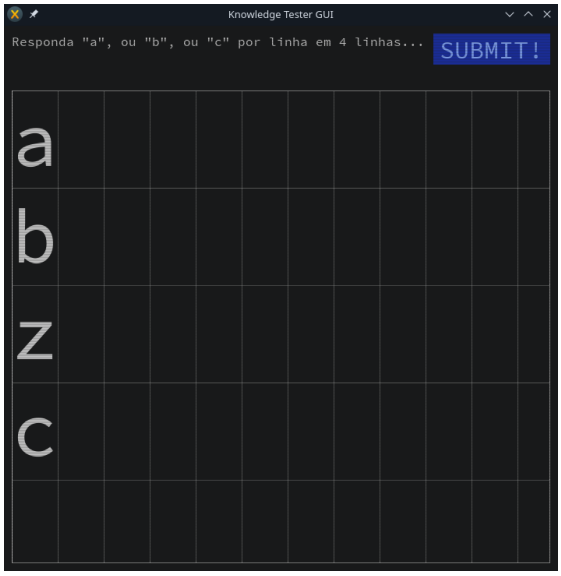
É de relembrar, conforme mencionado na secção sobre as *tags* que se podem usar neste *software*, que existe a *tag* “*[||]*” que pode ser usada para se aceitarem múltiplas alternativas por cada linha das respostas, inclusivé em multi-linha, o que significa que uma resposta de 10 linhas, pode ter em cada uma dessas dez linhas, várias respostas possíveis por linha.

Vejamos um exemplo definido por mim agora para exemplificar:



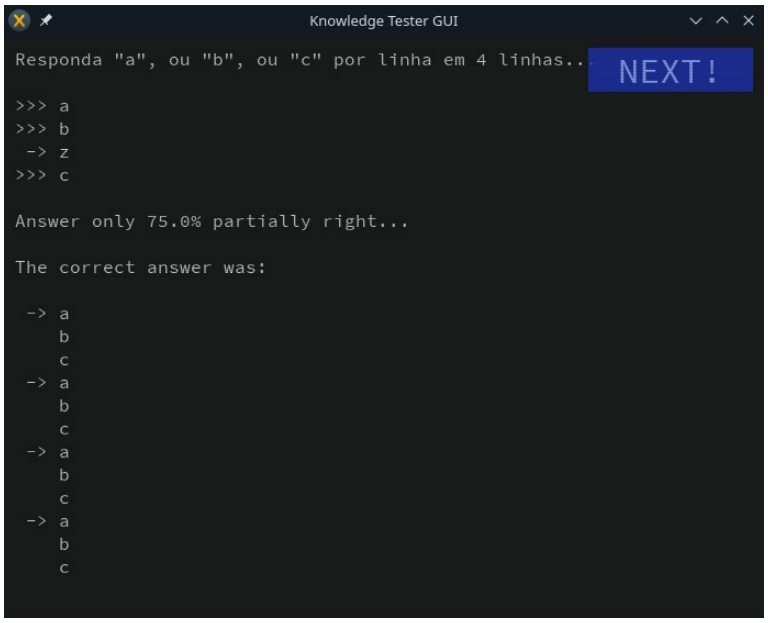
No exemplo acima, criei uma pergunta que me pede 4 linhas como resposta, e em cada uma das 4 linhas, posso escrever simplesmente “a”, “b”, ou “c”.

Vejamos o que respondo:



Como podemos ver acima, respondi errado apenas na terceira linha.

Vejamos agora o *feedback* dado:



Como podemos ver acima, o *software*, marcou como correctas a primeira, segunda e quarta opções, e errada a terceira, pois na terceira não respondemos qualquer uma das três escolhas válidas: as letras “a”, “b” e “c”.

Além de nos avisar qual falhámos, por termos apenas um “->” e não um “>>>” antes da linha correspondente na resposta dada, e nos alertar que assim só tivemos 75% da pergunta certa, ele mostra-nos quais seriam as respostas válidas, onde podemos ver que em cada linha temos a mesma iniciada por “->” e três sub-linhas com as tais respostas que poderíamos ter dado.

Penalizações nas respostas por linhas a mais

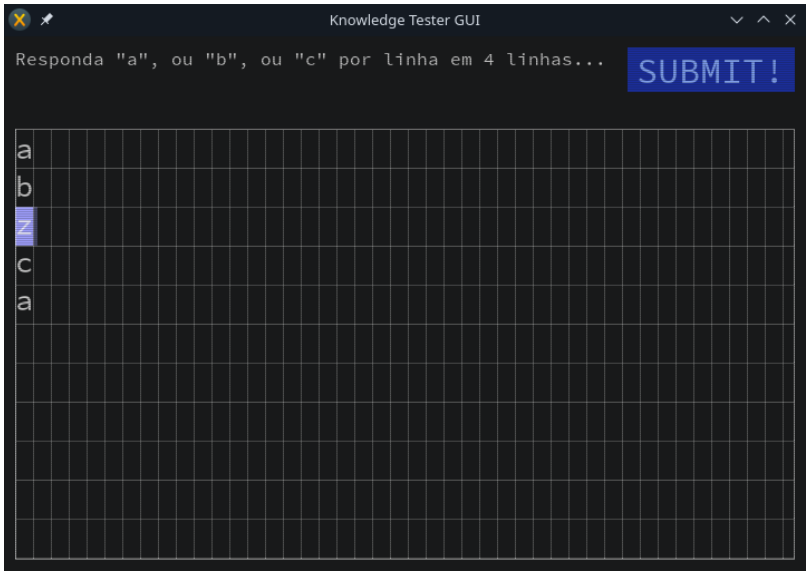
Não teria sentido, se as linhas válidas eram no exemplo acima, três letras (“a”, “b” ou “c”), que deixássemos o utilizador escrever muitas linhas na esperança de acertar algumas, senão o utilizador escreveria todas as letras de “a” a “z” e acertaria nas quatro válidas.

Também não seria completamente justo, em especial para quem faz exames ou pratica laboratórios, como por exemplo de redes (tipo configurações num *Cisco IOS*), cortarmos uma linha totalmente só por o utilizador a escrever mal numa e bem na seguinte, como por exemplo, pedirmos um “*conf t*” e o utilizador escrever “*conf*” e depois “*conf t*” (façam-me a vontade, imaginem que fez o “*conf*” sem querer, sem reparar, e estarmos a marcar ambas erradas quando uma estava certa.

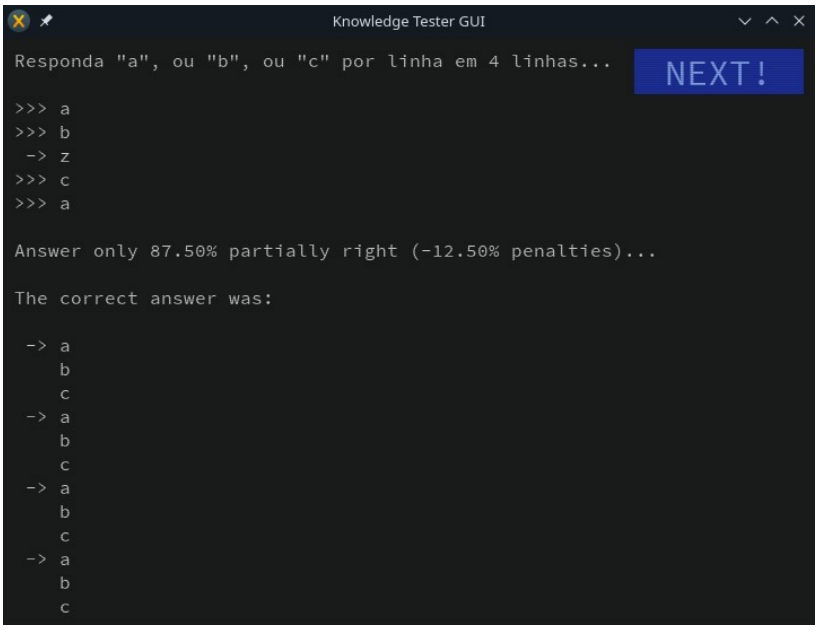
Neste caso, não marco essa alínea por defeito certa, nem errada, faço o meio termo, dou 50% da mesma como certa.

Por isso meti por defeito o *software* a aceitar como meio-certa qualquer resposta dada em duas linhas em vez de uma, pois parecia-me um desconto justo.

Vejamos um exemplo. Abaixo, vamos finjamos que não temos a certeza se a terceira linha seria um “z” ou um “c”:



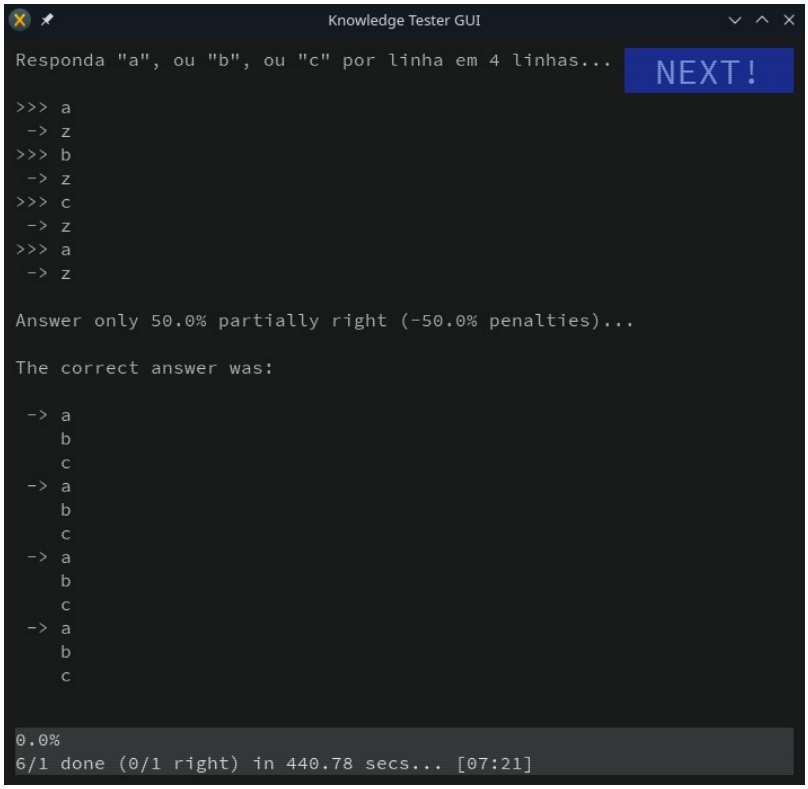
O *software* contará as outras três como correctas, e a tal que tinha duas linhas escritas, só verá uma delas como válida, e a valer metade:



Como podemos ver acima, como cada uma valeria 25%, a que vale metade, passa a valer 12,5%.

Vamos simular outro exemplo. Abaixo, vamos escrever duas linhas por cada uma das pedidas, todas elas com a letra certa, e com uma errada:

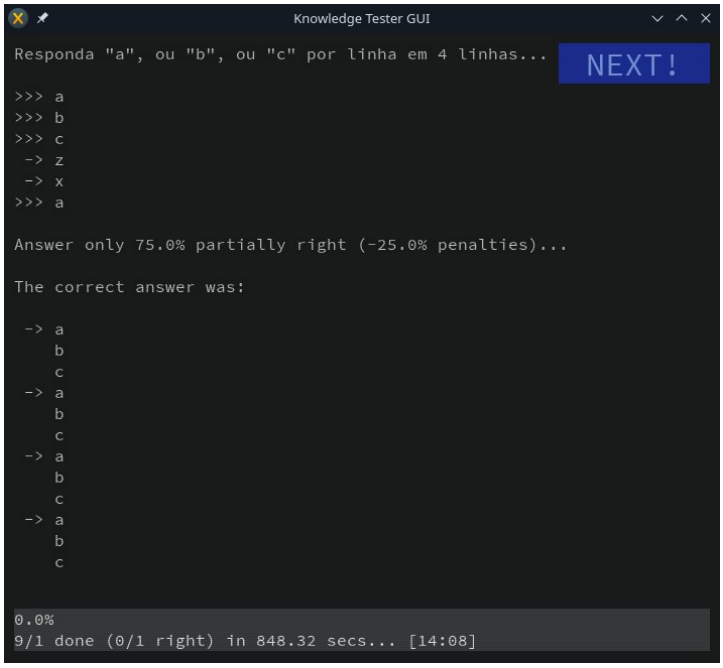
E como é óbvio, cada uma valerá metade, não os 25% do total da questão, mas 12,5%:



E se alguém decidir meter três opções por linha???

Isso já é ir demasiado longe. A partir do momento em que uma linha, tenha três tentativas, passará a ser um desconto de 12,5%, pois se cada uma vale 25%, e em vez de uma linha temos três, cada uma das duas linhas extra descontando metade, ou seja 12,5%, os tais 25% são anulados por 25% de penalizações:

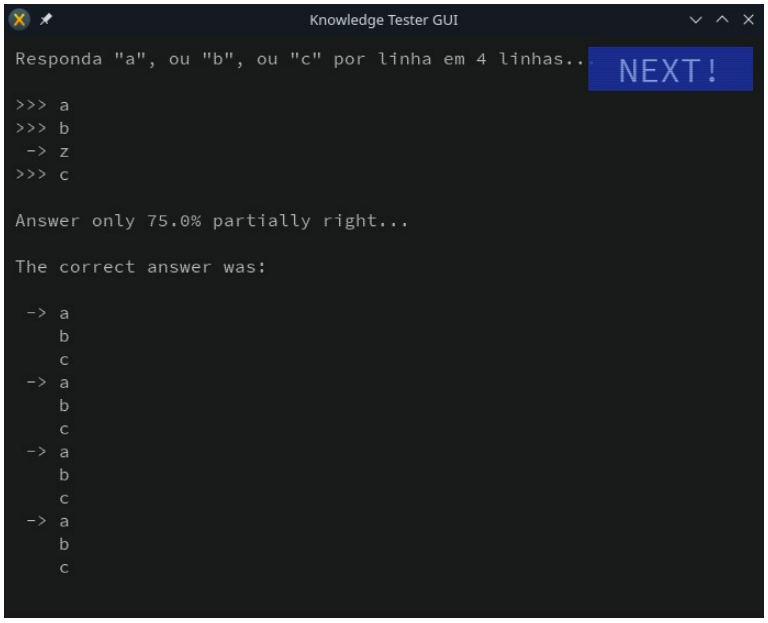
Acima temos assim a última linha com 3 possíveis, vejamos o resultado:



Tal como previsto, a última linha, teve duas (valendo metade cada uma) a descontar, por isso foi totalmente anulada, pois o seu valor de 25% foi anulado pelos 25% de penalizações, ficando assim a valer 0%.

Mas não façamos confusão, estes descontos só acontecem se o número de linhas de resposta, ultrapassar o número de linhas de resposta esperado.

Vejamos o exemplo que já foi dado acima:



Como podemos ver, ao falhar uma linha, ela simplesmente não conta, pelo que nenhuma penalização existirá.

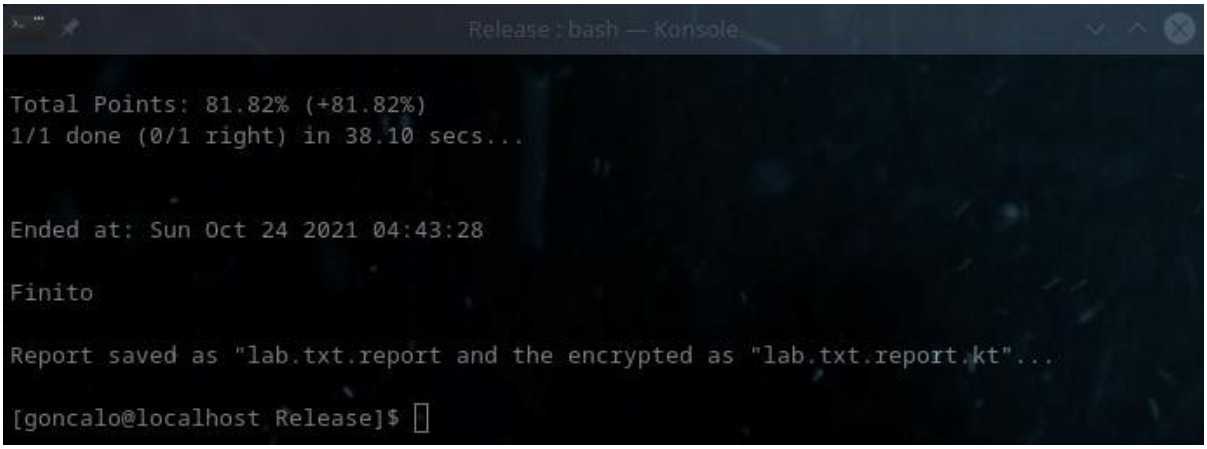
Mais tarde, em futuras versões (nunca antes de 2025), outras opções serão adicionadas, como associação de opções entre valores à direita e à esquerda, escolha múltima (múltiplas opções certas por pergunta), etc.

Até lá, vou andar ocupado com projectos mais importantes, e dado que este *software* é suficiente como ferramenta de trabalho para mim agora (eu crio e uso as minhas próprias ferramentas, ao meu gosto, como sempre gostei de fazer ao longo da vida), mantê-lo-ei como está.

Os relatórios dos exames

Os relatórios dos exames são extremamente importantes em avaliações.

Não teria sentido alguém ser avaliado sem um relatório exibindo todas as perguntas respondidas, não só porque o formador tem de rever as perguntas falhadas que podem ser erros de escrita, ou pequenas trocas mas onde o estudante mostrou conhecer a lógica por detrás do que queria fazer, mas também para evitar que notas e respostas sejam adulteradas por algum examinado, e em especial aquando de formações à distância:

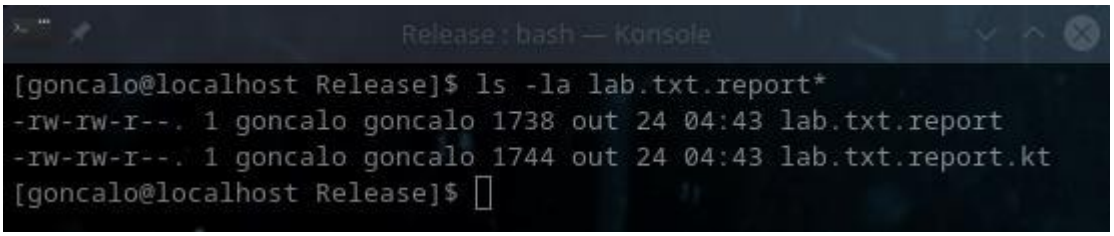


Ou seja, não basta o *software* nos dar uma nota automática assim que cada pergunta é respondida, o que é uma das enormes vantagens deste *software*, que é a correcção em tempo real do exame, e o examinado saber em tempo real que nota tem, se já atingiu os 50%, etc.

Sendo assim, sempre que é terminado um exame, são gerados dois tipos de relatórios diferentes, um ficheiro com o nome do exame e extensão "*.report*", e outro com o nome do exame e extensão "*.report.kt*", sendo que o primeiro está em texto bruto ("*raw*"), legível por todos, e que permite ao examinado ficar com uma cópia para ver onde falhou e onde tem de melhorar no Futuro, ou até saber que deve dizer ao formador que numa pergunta qualquer errou por algum motivo, e a segunda encriptada, onde apenas o examinador conseguirá

desencriptar e ter acesso ao relatório.

Ambos devem ser enviadas ao formador. O encriptado porque é a válido e inadulterável, e o de texto *raw*, só para a eventualidade de acontecer algo de errado, ou algum outro motivo:



Como podem ver acima, há dois tipos de relatórios diferentes, e o de extensão "*.report.kt*" é encriptado e por isso tem uns bytes a mais, típico do processo de encriptação usado neste caso, um sistema de encriptação baseado no algoritmo *Rijndael*, conhecido hoje em dia por *AES 256*.

Abaixo podem ver o conteúdo de ambos, e poderão reparar que o que está em modo de texto é visível ao aluno, que poderá levar para casa para ver onde falhou e acertou, e que tem literalmente todo o conteúdo que foi apresentado no ecrã durante o exame, sem excepção, tendo até o menu de entrada, opções escolhidas, perguntas e respostas, pontuação, mesmo tudo. Tudo o que foi apresentado no ecrã durante o exame, estará lá contido.

Abaixo podem espreitar o conteúdo dos ficheiros:

E como podem ver acima, um está em modo de texto *raw*, e o outro encriptado, pelo que contém código binário.

Os relatórios entre perguntas

Surgiu com o tempo a necessidade de serem gravados relatórios em tempo real, entre cada pergunta, derivado de alguns alunos terem a sua máquina a *crashar*, ou o portátil ficar sem bateria, ou o seu sistema *Windows* como de costume *crashar* e a máquina virtual *Linux* onde corriam o exame cair, ou a máquina virtual cair, ou terem aberto a mesma com tão pouca memória que deixaria de responder, etc.

Enfim, uma multitude de situações que vi acontecer e que me fizeram criar esta funcionalidade, para nesses casos, saber em que ponto o aluno ia, e poder juntar essa nota ao resto do exame que recomeçariam, cruzar perguntas, extrapolar, etc.

Sendo assim, agora, sempre que uma pergunta é respondida, é gravado em tempo real um relatório na pasta onde o exame é corrido, com a última pergunta e resposta respondida, para termos acesso apenas a ela e à nota do aluno, o tempo que demorou, quantas perguntas saltou, quanta percentagem de nota tinha, quantas tinha tentado, etc.

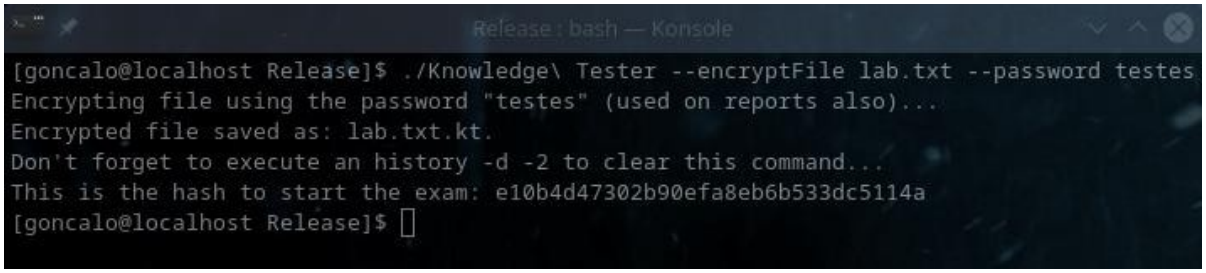
Salvo erro, pois não tenho a certeza, por ter largado este *software* e esta funcionalidade há meses, penso que este relatório ou sai encriptado para que o aluno não possa ver em tempo real quais as respostas que teve, nem enviar a outros, ou então em modo de texto *raw*, sem encriptação, mas com apenas a última pergunta, para evitar que faça o mesmo.

Um dia actualizarei aqui qual delas é a correcta, fica para 2025.

Exames Protegidos

Como proteger um exame

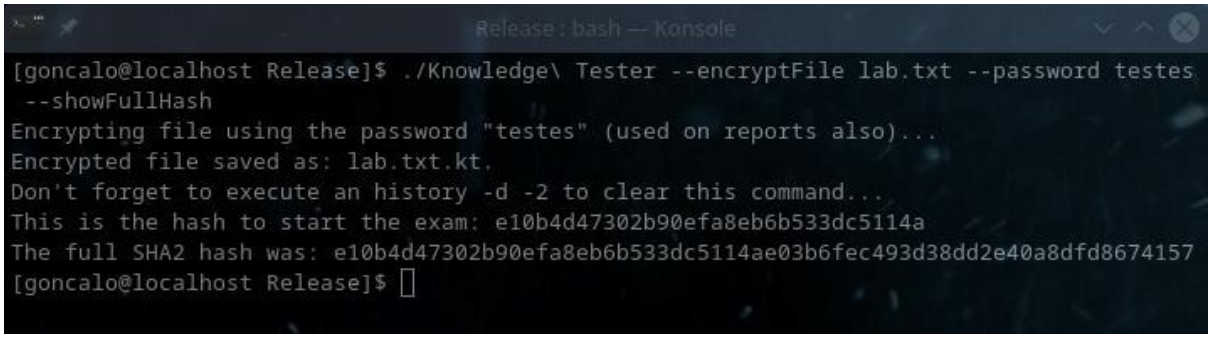
Assim que criamos um exame, podemos encriptar o mesmo com o parâmetro "*-- encryptFile*" seguido do nome do ficheiro, junto com o parâmetro "*--password*" seguido da password que pretendemos usar para a encriptação:



Escusado será dizer que para termos protecção total, devemos usar passwords com muito mais caracteres que esta, com símbolos à mistura e não apenas letras, e que não contenham só palavras de dicionário como a "testes" usada acima, pois se alguém tentar um ataque inicial de *brute-force* a este ficheiro encriptado, e experimentar fazê-lo com um dicionário

pequeno onde esteja a palavra “testes” contida, ele poderia ser desencriptado não em milhões de anos mas em minutos ou horas.

Pode ser usado também o parâmetro “*--showFullHash*” durante a encriptação, para visualizarem a *hash* inteira original, ao invés de apenas a cortada, e tal será falado posteriormente neste manual:



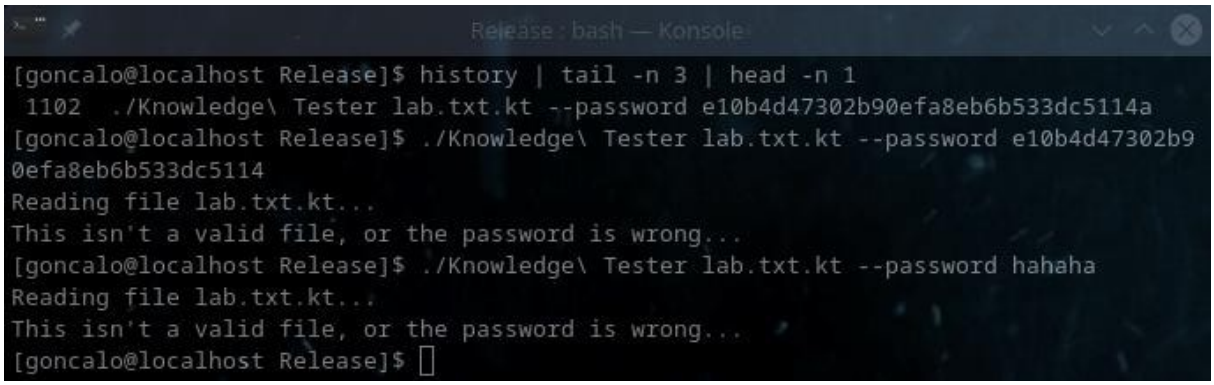
Falamos mais sobre a força das *passwords* no tópico seguinte mais abaixo.

Como correr um exame protegido

Notem na imagem anterior que nos é dada uma hash para se poder iniciar o exame, que no exemplo acima é “*e10b4d47302b90efa8eb6b533dc5114a*”, e que será a mesma se tentarem encriptar alguma coisa com a *password* “testes” nas vossas máquinas também, e que é encurtada a 32 caracteres para ser mais fácil de transmitir, mas que não deixa de ser mais do que suficientemente segura.

Esta é a password que deverão dar aos formandos/alunos para eles iniciarem os exames. Desta forma eles conseguem iniciar o exame com a password dada, mas não conseguem desencriptar os relatórios com a mesma.

Vejamos como se corre um exame com essa password:



No exemplo acima podem ver que corri antes (sem problemas) o exame da maneira correcta, com a palavra-passe que me foi dada inicialmente ao encriptar o exame (a "*e10b4d47302b90efa8eb6b533dc5114a*"), e correu sem sucesso, e é importante referir que qualquer tipo de *hash* usada por este *software* tem força mínima de *SHA2* e devidamente "salteada" para evitar ser descoberta a origem através de *rainbow tables*.

Mas de seguida cortei apenas um caractere a essa palavra-passe (o "*a*" final), e já falhou, com a mensagem "*This isn't a valid file, or the password is wrong…*", e se tentarmos também outra palavra qualquer, como o "*hahaha*" acima, falhará também.
O exame só é aberto na hora certa, quando o formador assim quiser (quando lhes der a *password*), e com a *password* certa.

O exame também corre com a *password* original usada pelo formador, ou seja, o formador pode usar a *password* "testes" e dá-la aos alunos e eles correrem o exame com ela, desta forma todos sabem a *password* original dentro da turma, mas fora da turma ninguém a saberá. O único problema é que desta forma, qualquer aluno poderá adulterar um ficheiro de relatório de exames, apesar de ter de saber fazê-lo.

Há uma opção de fazer com que os exames encriptados com *password* possam ser executados/abertos sem qualquer *password*, que será explicado abaixo.

Como fazer exames encriptados abrir sem *password*?

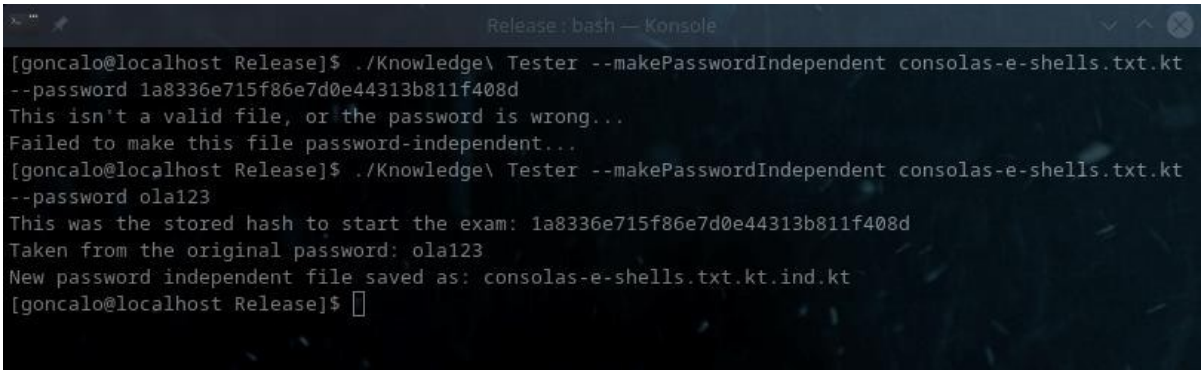
Com a opção "*--makePasswordIndependent*", podemos fazer com que um teste/exame encriptado com *password*, possa ser aberto sem qualquer necessidade de uso de *password*.

Ele continuará encriptado com a *password* usada originalmente, o que significa que exames executados com esse ficheiro, gerarão relatórios encriptados que só poderão ser visualizados/desencriptados com a *password* original (e não a dada aos formandos para execução), e significa também que o exame em si só poderá ser também desencriptado na mesma com a *password* original, que nunca será guardada no ficheiro em si.

Mesmo assim, bastar-nos-á executar "*./Knowledge\ Tester ficheiro.txt.kt*" para o exame abrir, sem ter de saber a *password* original.

Isto é útil quando temos um exame para dar a formandos, que não podem desencriptar para ver as suas perguntas e respostas, nem para verem os relatórios gerados, mas ao mesmo tempo não queremos obrigá-los a ter de colocar *passwords* para aceder.

Assim basta executarem e já está, sem deixar de estar protegido:



Acima podemos ver que só com a *password* original se pode executar esta operação, senão obviamente ganhava-se acesso ao ficheiro mesmo sem ter a mesma.

O ficheiro final acaba por ter adicionada a extensão "*.ind.kt*", de "independente", que poderá ser obviamente alterada mais tarde para um nome mais fácil de memorizar ou usar.
A partir desta operação nunca mais será pedida uma *password* para se abrir este ficheiro.

Atenção que depois, tal como está feito o *software* de momento, não podemos forçar argumentos ou outras operações neste ficheiro, por ser já *stand-alone*, sendo que tais operações têm de ser feitas antes de se tornar *stand-alone*, como por exemplo a "*--forceArguments*" ou algumas outras.

Não fica gravada no próprio exame a *password* original?

Não. O sistema de encriptação deste *software* foi pensado de forma a que um exame quando é corrido, saiba precisamente como gerar um relatório que possa ser desencriptado apenas com a *password* original, sem nunca a saber sequer.

Isto pode parecer confuso a quem nunca programou algo com encriptações, mas é possível.
Isto significa que o relatório tem informação suficiente para gerar um relatório que só possa ser aberto com a *password* "testes", mas ninguém conseguirá nunca encontrar nesse relatório a própria *password* "testes".

Escapa ao âmbito deste manual explicar como tal é feito, mas pode-se garantir que a *password* original usada pelo formador para gerar um exame, nunca sairá das suas mãos a qualquer momento, e só com a mesma é que um relatório poderá ser desencriptado.

É importante referir assim, que essa *password* usada pelo formador para gerar exames, nunca é guardada em lado nenhum, nem sequer no exame, nem noutro lado qualquer, a nenhum momento, sendo que se o formador a usar, e correr o comando "*history -d -2*" logo de seguida para eliminar a mesma do histórico da *shell Linux* que está a usar, ela desaparece de vez e ficará apenas guardada na sua mente.

Que *password* têm exames encriptados sem *password*?

Um exame encriptado sem o uso do parâmetro "*--password*" seguido da palavra-passe, poderá ser aberto sem o uso de qualquer tipo de *password*, ele estará encriptado com a *password* por defeito do *Knowledge Tester*, e por isso ao ser aberto sem *password* será essa a que ele irá tentar usar para desencriptar o ficheiro, e com sucesso.

Mas este tipo de operação não tem grandes vantagens, apesar de ser possível executá-la.

Cuidados a ter quando se encripta um ficheiro de exames

Podem ver acima que ao encriptarmos o ficheiro, nos é dado o nome do ficheiro já encriptado (no exemplo acima "*lab.txt.kt*", e um pequeno alerta de que devemos usar o comando "*history -d -2*" para limpar o comando introduzido, pois se não o fizermos, qualquer pessoa com acesso à máquina onde o exame tenha sido encriptado, pode visualizar a lista de comandos introduzidos, e ver a *password* usada. Com esse comando "*history -d -2*", o comando é apagado, e a *password* desaparece assim e passará a estar guardada apenas na mente de quem criou o exame.

Não se deve encriptar o exame também em máquinas não fidedignas, pois podem conter algum tipo de *key-logger* e assim a *password* ser acedida por terceiros na mesma, ou enviá-las por email ou de outra forma na *Internet* em redes não fidedignas (pode ser vítima de *sniffing* ou outro estratagema), etc.

O poder da encriptação *AES 256* e que *passwords* usar

Parte da força da encriptação reside na palavra-passe usada, e não apenas no algoritmo. Quanto mais caracteres tiver a palavra passe, mais forte fica.

Pois se cada caractere pode ter 256 valores diferentes (2 elevado a 8 bits), poderíamos ter 281.474.976.711.000 combinações de palavras-passes diferentes, ou até um pouco menos se contarmos apenas uns 128 bytes (os caracteres normalmente digitados mais uns extra), o que já dificultaria, se usássemos símbolos à mistura como "$&#)(!/)(" para forçar o uso desse tipo de *brute-force* e tentar invalidar assim ataques por dicionário.

Mas se usarmos uma palavra-passe como "testes", qualquer ataque de dicionário poderia facilmente descobrir a *password* em "pouco" tempo.

Por isso é importante usar *passwords* minimamente grandes, com símbolos diferentes, para assim tentarmos evitar o máximo possível ataques de dicionário, como por exemplo "oiuW()/"ioYU#98fh#"(hfoiah89"(9h", e isto usando 32 caracteres (para quê mais?).

A pessoa cria a *password* e guarda-a bem, ou então algo fácil de memorizar, como "Este! é"o#exame$da%turma&1975.", que como podemos ver, é fácil de memorizar, tem letras maiúsculas e minúsculas, vários símbolos (os das teclas numéricas com *shift* e seguidos, e até números tem, e seria uma palavra-passe mais do que forte.

Agora, qual o poder máximo de uma encriptação deste género?

Vamos colocar as coisas desta maneira, que é uma maneira engraçada que já vi ser usada antes:

Dizem que existem perto de uns 1.3*10^50 átomos no nosso planeta, na sua totalidade (uns 1.300.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000.000 átomos).

Se cada um destes átomos fosse um computador, e todos juntos estivessem a tentar 10.000 milhões de possíveis chaves (possíveis *passwords*) por segundo, mesmo assim, ainda levariam 2.840 milhões de anos todos juntos a testar todas as combinações possíveis para desencriptar o ficheiro. :-)

Não é possível quebrar uma *password* super forte encriptada com este algoritmo de encriptação (neste caso o *AES 256*) se a palavra passe for suficientemente boa, a única forma de o fazer, seria por métodos alternativos, seja descobrindo falhas não conhecidas ainda no algoritmo, ou sacando a palavra-passe ao formador/professor de alguma forma.

Agora se alguém usar a *password* "testes", ela poderá ser desencriptada em bastante menos tempo, como é óbvio. Mesmo assim usei uma pequena protecção extra, para dificultar a tarefa de desencriptação, mas não passa do velho método de adicionar fechaduras à porta da casa, ou seja, pode demover alguns assaltantes de a roubar, mas se algum estiver mesmo motivado, acabará por entrar na mesma.

Por isso, usar palavras-passes o melhores possível, pois um algoritmo de encriptação poderoso de nada serve se quem o utiliza usar *passwords* fracas, é o típico exemplo de uma corrente ser tão forte quanto o seu elo mais fraco.

Relembra-se que a encriptação usada é tão potente no dia de hoje (que posso dizer que vai além do simples algoritmo *AES 256*), que mesmo que usasse o algoritmo *AES 256* sem nada mais, necessitaríamos hoje em dia de milhões de milhões de anos para tentar desencriptar com métodos *brute-force* e tecnologia actual, levaríamos largas vezes muito mais tempo do que a própria idade do Universo.

É por isso um método seguro, e um formador ou professor pode literalmente enviar o exame aos seus formandos/alunos, um ano antes, dizendo "está aqui o exame que vão fazer daqui a

um ano", e dar-lhes a palavra-passe apenas um ano depois (para abrirem o exame), que por muito que tentem, nunca conseguirão adivinhar a palavra-passe, a não ser que a conseguissem sacar do próprio formador, claro está.

Isto torna o método seguro até para formações à distância.

Os ficheiros *.kt* e *.ind.kt*

Atenção que podem fazer com que um exame seja enviado aos examinados, sem terem de colocar a *password* para o executar, ou seja, sem terem de meter um *–password palavraPasse* ao arrancar o exame, se tiverem autenticação *web*, pois se tiverem autenticação *web*, eles só começam o exame quando e apenas quando for dada permissão, e por isso a *password* deixa de ser tão importante, mas existirá na mesma para evitar que desencriptem e voltem a encriptar o relatório encriptado no fim do exame (para evitar que adulterem o relatório).

Para isso só temos de usar o argumento já explicado noutra parte deste manual, que é o já conhecido *--makePasswordIndependent*, assim:

./Knowledge\ Tester --makePasswordIndependent exame.kt --password palavraPasse

Após isto, será gerado um ficheiro com um "*.ind*" adicionado ao fim do nome, ou seja: "*exame.kt.ind*", para que o original "*.kt*" não seja apagado no processo.

Por norma eu mudo-lhe o nome para "*exame.kt*" no fim, e envio-o assim. E ele tem já a *password* embebida (mas não na realidade, ou seja, não dá para ser recuperada por *reverse engineering*), os exames já começarão sem *password* dada, por ser "*password independent*", e a autenticação como será feita via *web*, só começam o exame se autorizarmos no nosso painel de controlo remoto.

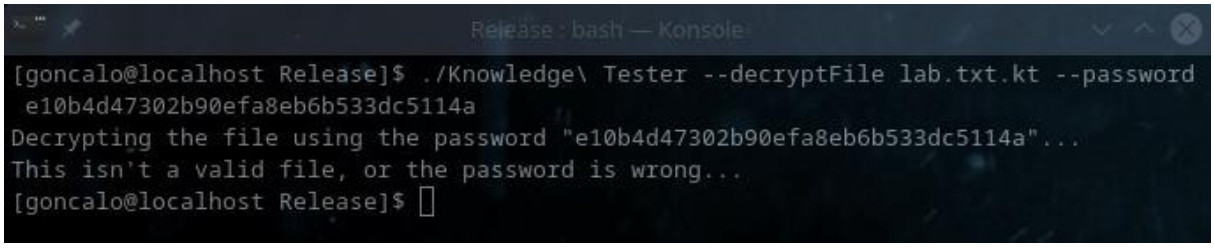
Atenção que pode ser usada esta funcionalidade também, quando queremos distribuir exercícios pelos alunos, para praticarem (e não apenas exames em si), mas não desejamos que possam desencriptá-los para os adulterar.

Nesse caso, metemos uma *password* neles, para que não possam ser desencriptados e adulterados, mas como está embebida no exame, ela nunca é pedida, serão simplesmente exames encriptados mas sem necessidade de *password* para serem corridos, e assim evitamos que possam ser extraídos/desencriptados/adulterados, e distribuídos com segurança.

Como desproteger um exame

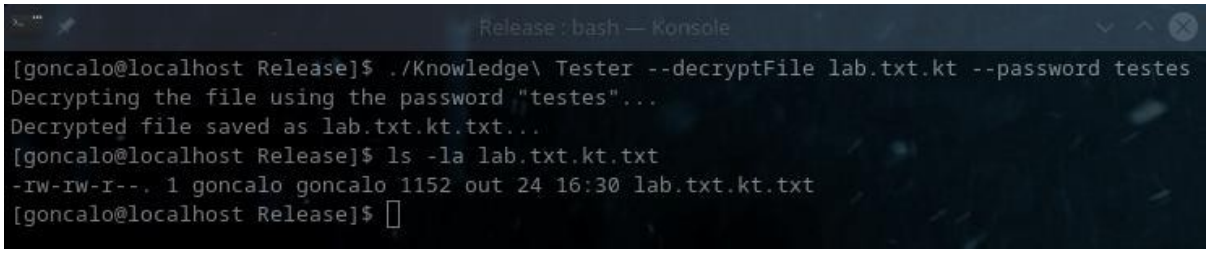
Para desproteger um exame, só necessitamos de ter a *password* usada originalmente para o criar, e usar o parâmetro "*--decryptFile*" seguido do nome do ficheiro, e o "*--password*" seguido da *password* original usada para o encriptar inicialmente.

No exemplo abaixo podemos ver o que poderia ser um formando a tentar desencriptar o exame usando a *password* que é dada para os formandos abrirem o exame, e podem ver que falharia:



Isto porque tal como foi explicado antes, só mesmo com a *password* original, neste caso a "testes", poderá ser desencriptado um exame.

Vamos abaixo ver que com um simples "*Knowledge\ Tester –decryptFile lab.txt.kt –password testes*", o exame é desencriptado, e é gerado um novo ficheiro com o conteúdo desencriptado, adicionando-se um "*.txt*" à extensão original:

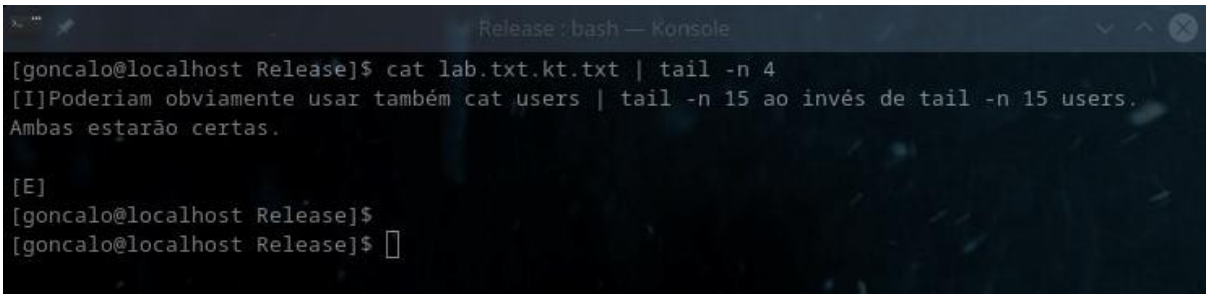


Como podem ver pela imagem acima, o ficheiro "*lab.txt.kt.txt*" foi criado, e contém o conteúdo desencriptado do exame original, o que pode ser útil caso o formador tenha perdido o exame original em texto e necessite de o recuperar.

Pode é ser algo confuso o sistema adoptado, de adicionar "*.txt*" quando desencriptamos, ao nome do ficheiro, e depois "*.kt*" quando o encriptamos novamente, e podemos acabar um dia com um ficheiro de nome "*lab.txt.kt.txt.kt.txt.kt.txt.kt*" se o fizermos de forma indefinida, mas isso só acontece se a pessoa o quiser, pois o formador tem apenas de mudar o nome para o desejado com um simples "*mv lab.txt.kt.txt lab.txt*" e tudo fica perfeito.

Lembrem-se que depois de desencriptado, o ficheiro continuará a não ter qualquer referência à *password* original, ela nunca é escrita nem contida em lado nenhum.

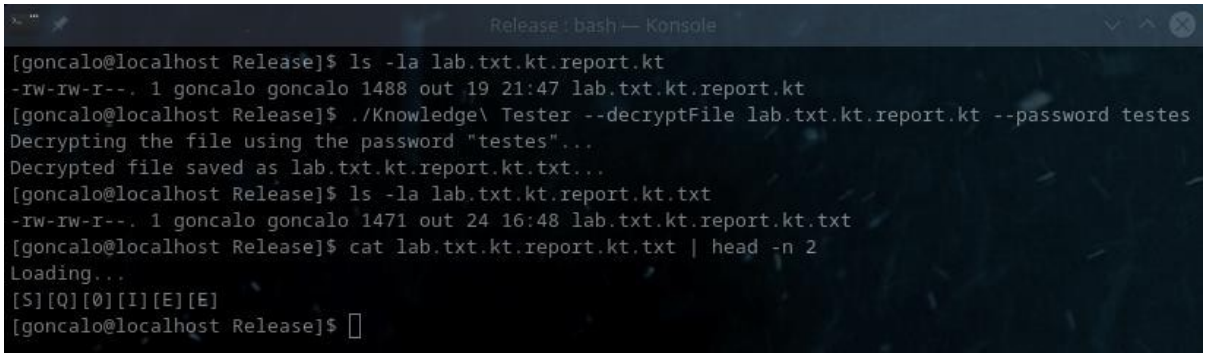
Podemos ver abaixo o conteúdo do ficheiro de exame já desencriptado, para verem que está em texto legível já:



Como visualizar o conteúdo de um relatório encriptado

Com os relatórios encriptados, o método é o mesmo do usado para desencriptar exames encriptados, ou seja, corremos o "*Knowledge\ Tester*" seguido de "*--decryptFile*" e nome do ficheiro, e um "*--password*" seguido da *password* usada para criar o exame original, só que ao invés de estarmos a desencriptar um exame, estaremos a desencriptar um relatório gerado por um exame, mas o funcionamento por detrás é o mesmo, são desencriptações aos ficheiros.

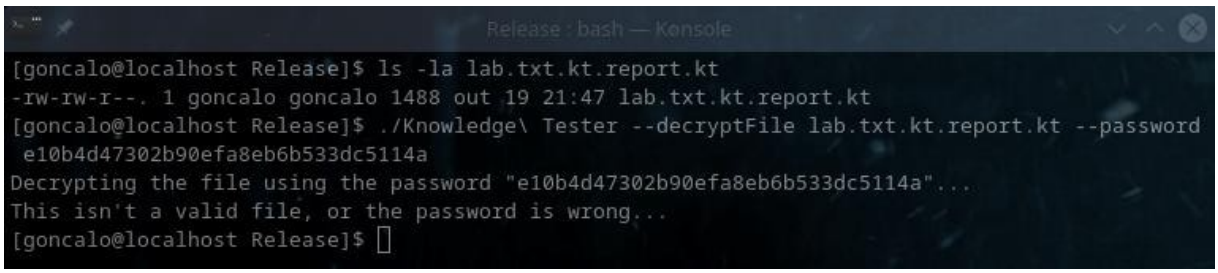
Abaixo fica um exemplo da desencriptação de um relatório gerado por um exame:



Como podem ver pela imagem acima, o ficheiro "*lab.txt.kt.report.kt*" foi desencriptado (nome confuso não é?), e o seu conteúdo passou a estar em modo de texto "*raw*", legível.

Depois cabe ao formador mudar o nome para o desejado.

Escusado será dizer que a *password* dada aos alunos para desencriptar, a "*e10b4d47302b90efa8eb6b533dc5114a*", não resultaria, pois assim qualquer aluno poderia desencriptar o relatório e alterá-lo sem problemas:



Como podem ver acima, a *password* dada aos estudantes, não desencripta o relatório, pelo que ao ser usada recebemos um erro de "*This isn't a valid file, or the password is wrong…*", e neste caso sabemos que é a *password* que está errada, pois o ficheiro é mais do que válido e já foi desencriptado antes com a *password* correcta.

Como forçar opções num exame

Podemos forçar opções num exame, fazendo com que estejam já embutidas no mesmo, e evitando que sejam alteradas à mão na linha de comandos, com um argumento específico, o "*-- forceArguments*", seguido do nome do ficheiro.

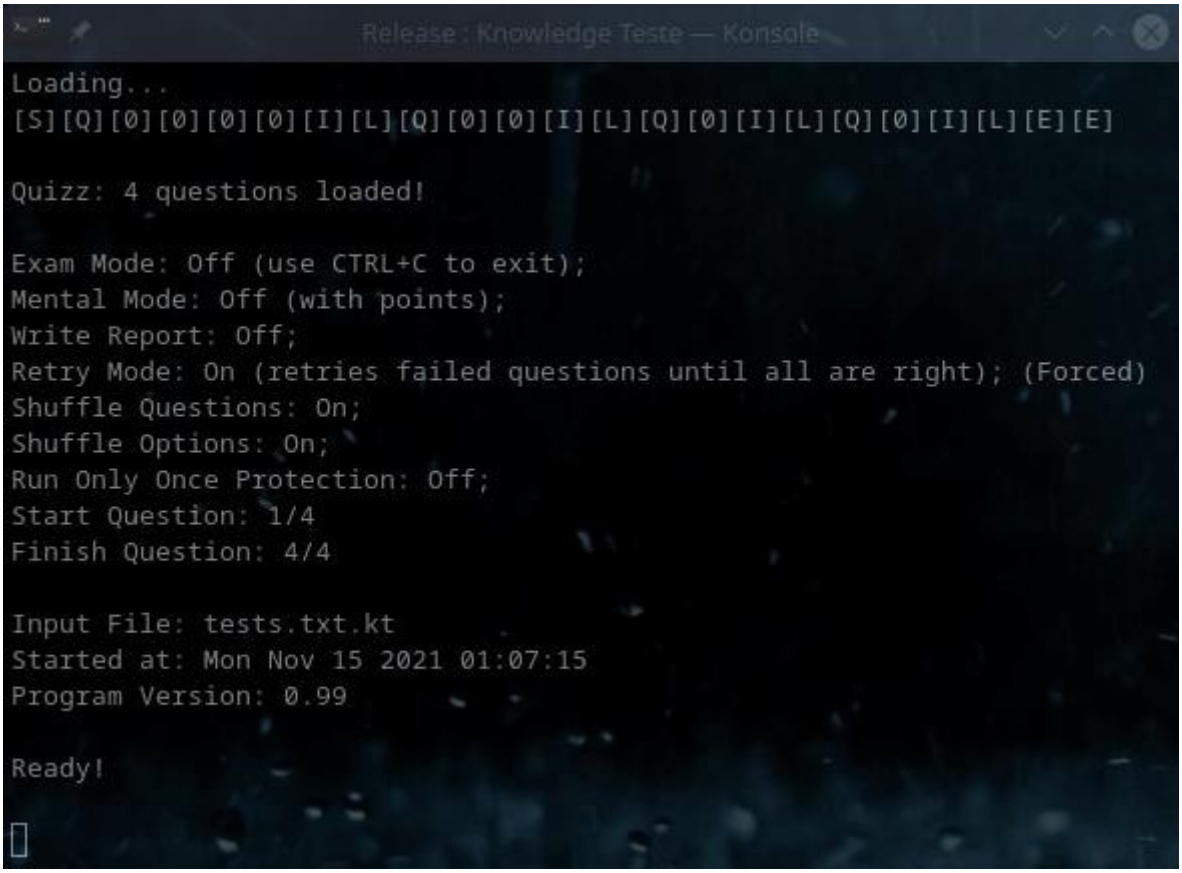
Por exemplo, se adicionarmos o argumento "*--forceArguments*" seguido do nome do ficheiro, e colocarmos o argumento ou argumentos em questão, esse ficheiro será encriptado e com as opções já embutidas, sem permitir que o utilizador final as altere.

Como exemplo, podemos tentar um "*Knowledge\ Tester --forceArguments ficheiro.txt --retryOff --password hello*", para forçar o argumento "*--retryOff*" como estando sempre activado, e mesmo que tentemos arrancar o exame com um argumento "*--retryOff 0*" (para o desligar), ele estará sempre ligado na mesma.

É importante definir a *password* para que só através dessa *password* original consigamos desproteger o exame e remover a opção.

É importante reparar que num "*--retryOff*" ou "*--shuffleOptionsOff*", que por defeito desligam essas funcionalidades, podemos adicionar um "*0*" à frente, e assim elas têm o efeito contrário, ligando as mesmas.

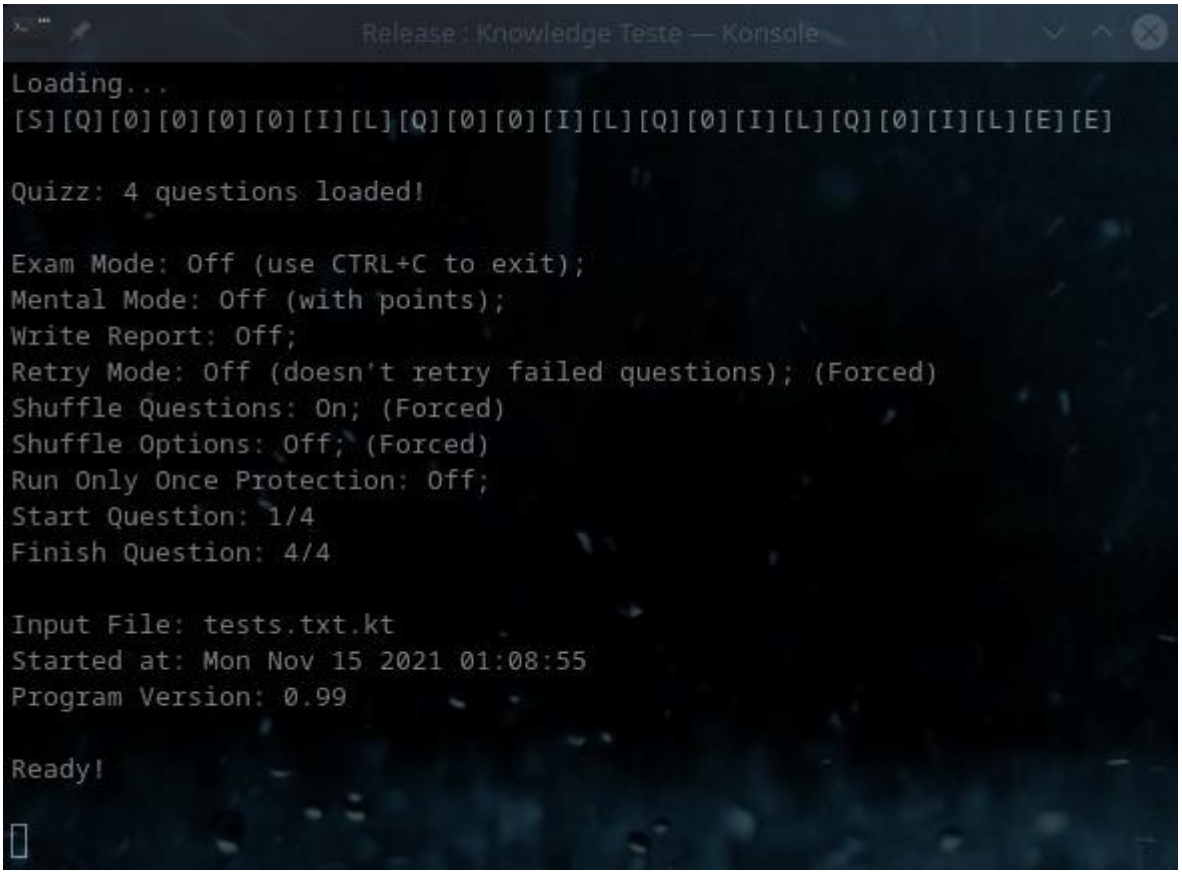
Assim por defeito elas têm um valor de "*1*", mas podemos forçar um valor negativo, como "*0*":



No exemplo acima, o "*Retry Mode*", definido pelo argumento "*--retryOff*", está definido como sendo "*0*", ou seja, o modo de "*Retry*" está definido como estando sempre ligado, daí ter um "*(Forced)*" à frente no ecrã de entrada inicial.

Se não tivessemos usado o "*0*" à frente do argumento "*--retryOff*", teríamos o "*--retryOff*" como sendo verdadeiro, e aí nunca estaria ligado o "*Retry Mode*", nem que o tentássemos ligar à mão.

No exemplo abaixo, temos esse modo invertido, sempre ligado o "*--retryOff*", e com isso, sempre desligado o "*Retry Mode*", e ainda forçámos alguns outros parâmetros, com a linha de comandos "*./Knowledge\ Tester --forceArguments tests.txt --retryOff 1 --password hello --shuffleOptionsOff 1 --shuffleQuestionsOff 0*":



Podemos ver acima assim vários parâmetros forçados (com um "*(Forced)*" à sua frente), sendo que esses parâmetros não podem ser alterados através da linha de comandos, pelo que só a pessoa com a *password* original os poderá alterar, o que é muito útil em caso de exames em que desejamos que os formandos não consigam alterar certas opções.

Como preparar um exame pronto para ser distribuído

É super importante, como é óbvio, sabermos preparar um exame para o podermos enviar aos alunos para eles o executarem, pois sem isso, teríamos de enviar o exame em modo *raw*, com o texto em bruto, etc.

Aqui vamos aprender como forçar opções durante a execução do exame, como o encriptar, entre outras coisas.

Imaginemos que vamos criar um exame sobre o ficheiro exame.txt…

Só temos no fundo de invocar o *software* na linha de comandos, mas em vez de usarmos um parâmetro como "*--encryptFile*", usamos o "*--forceArguments*", que fará a mesma coisa, mas com argumentos forçados, assim (irei colocar algumas opções ao calhas):

./Knowledge\ Tester --forceArguments exame.txt --shuffleQuestionsOff 0
--shuffleOptionsOff 0
--writeReport 1
--minVersion 110
--runOnlyOnce 0
--terminal 0
--examMode 1
--retryOff 1
--mentalMode 0
--password "ThreeSuperDucksWereFlyingUpToTheInfinity"

Descrição dos argumentos para forçar opções

Fica a descrição de cada parte da linha acima:

- ***./Knowledge\ Tester*** - invoca o programa;

- ***--forceArguments*** **exame.txt** – diz que iremos encriptar o ficheiro "exame.txt", forçando argumentos que estarão activados quando se executar o exame;

- ***--shuffleQuestionsOff*** **0** – Esta opção sem o 0 a seguir, seria por defeito 1, ou seja: *true*, e por isso temos de colocar um 0 para a desactivar.

  E se desactivamos o desligar do tornar as questões aleatórias, significa que estamos a colocá-las como aleatórias;

- ***--shuffleOptionsOff 0*** – Neste caso, estamos a dizer ao *software* que desejamos que as opções sejam exibidas em cada pergunta de forma aleatória;

- ***--writeReport*** **1** – Aqui pedimos ao *software* que guarde um relatório no fim do exame, para o podermos corrigir se necessário;

- ***--minVersion*** **110** – É importantíssimo este parâmetro, pois se não o usarmos, algum estudante pode correr o exame enviado, numa versão antiga do *Knowledge Tester* onde as protecções ainda não existiam, e assim conseguir fazer batota, ou algumas funcionalidades requeridas pelo exame não funcionarem, por isso devemos sempre colocar a versão mínima de execução, que sem ela, o exame não será executado.

  Atenção que 110 significa 1.10, por isso se a versão for 1.10, colocamos 110, e se for 0.90, colocamos 90, é sempre 100 vezes superior;

- ***--runOnlyOnce 0*** – Esta é uma protecção criada inicialmente para não permitir que o mesmo exame fosse executado duas vezes na mesma máquina, para evitar que alunos ao falhar algumas perguntas, cancelassem o exame e o reiniciassem, sabendo já as respostas.

- Com esta opção, a mesma máquina *Linux*, nunca mais voltaria a deixar executar o mesmo exame, dizendo que já não poderia executar aquele exame mais vezes nessa máquina.

  Posteriormente foi adicionada a autenticação remota, porque era possível aos alunos se estivessem em casa, criar máquinas virtuais diferentes, e falsificar exames desta forma, e com a autenticação remota isso já não acontecia.

  Devido a isso, se usarmos a autenticação remota, podemos e provavelmente devemos, meter um *–runOnlyOnce 0* para desligar esta funcionalidade, porque senão só com a *password* é que poderíamos anular essa protecção da máquina do aluno remotamente caso a máquina do mesmo fosse abaixo (por exemplo com uma falha de energia), o que não seria viável;

- ***--terminal 0*** – Esta opção deve ser usada em modo de exame, pois é comum em exames remotos, os alunos falarem entre eles em aplicações de *chat* como o *Discord* (mais usado pelos mais jovens), o *WhatsApp* (muito usado hoje em dia, em especial pelos adultos), ou outros como *Telegram*, etc, chegando alguns a cair no desplante de o fazer nos próprios *chats* da aplicação *Teams* da própria instituição onde tiram o curso!

  Mas o mais usado entre adultos é por norma o *WhatsApp* hoje em dia.

  E nestes casos, os alunos conseguem facilmente num terminal de *Linux* seleccionar o texto de uma pergunta, pesquisá-lo num *chat* de *WhatsApp* onde outros possam já ter respondido à mesma, e assim ajudarem-se uns aos outros remotamente, algo que tentam fazer sempre em grupo em exames remotos.

  A maneira de combater isto, é limitar o tempo do exame, com bastantes perguntas, e correndo a aplicação em modo gráfico (onde não dá para copiar texto da mesma propositadamente (só se pode copiar e colar dentro da própria aplicação), e devido a isto, temos de fazer com que o argumento *–terminal 0* seja forçado, como “0”, ou seja, desligado, para nunca permitir aos alunos executar o exame em modo de terminal.

  Desta forma, não têm tempo para escrever a questão à mão num grupo de alunos para pedir ajuda, pois não responderiam às perguntas todos a tempo, daí ser tão necessária;

- ***--examMode 1*** – Esta obviamente diz ao *software* que irá correr o exame em modo de exame, com tudo o que isso acarreta, como relatórios guardados, não deixar sair com "*CTRL+C*" ou "*CTRL+W*", etc, mecanismos de detecção de *cheating* (nem todos descritos neste tutorial), etc.

- ***--retryOff* 1** – Aqui estamos a dizer que não permitimos que as perguntas voltem a ser questionadas, pois por defeito, o *software* volta a repetir as falhadas no fim do exame, e continua a repeti-las, até que o examinado as acerte a todas (quando em modo de prática e não de exame, claro está), o que faz com que o exame só termine quando a pessoa saiba as repostas todas de cabeça.

  Obviamente, num exame, não podemos permitir que isto aconteça, por isso temos de desligar o modo de "*retry*", daí metermos "1" de *true*, porque queremos que o modo de *retry* esteja mesmo "*off*";

- ***--mentalMode* 0** – Este é um modo de prática, e quando estamos em modo de exame ele é desligado automaticamente, mas pelo sim pelo não, meti-o aqui para perceberem como funciona, como o desligamos;

- ***--password* "*ThreeSuperDucksWereFlyingUpToTheInfinity*"** – Aqui a palavra-passe, e já sabem que quanto mais letras e caracteres melhor, mas aqui não meti especiais, só para testes, para verem como funciona;

- **Parâmetros vários de autenticação *web*** – Existem outros parâmetros super importantes para impedir alguma batota por parte dos formandos, mas que são descritos no capítulo seguinte sobre Autenticação *Web* e temos de os configurar senão permitimos aos formandos que remotamente possam fazer batota nos exames;

De seguida, vamos ver como configurar essa mesma autenticação remota num exame.

Autenticação via *Web*

Um dos problemas que tive ao começar a usar este programa para exames remotos, foi o evitar que os alunos pudessem correr exames em máquinas virtuais distintas, para tentar sacar respostas de um para as usar no outro.

Protecção para correr exames apenas uma vez

Foi criada inicialmente uma protecção, que era accionada pelo parâmetro –*runOnlyOnce*, mas essa funcionalidade era facilmente contornada pelos alunos, porque apesar de caso existisse, a aplicação não correr o exame novamente, nem que se reiniciasse a máquina (algo que alguém bom em *Linux* poderia levar algum tempo a descobrir), poderiam sempre criar uma máquina virtual nova e fazer o exame em separado. No fundo, sacavam o exame, duplicavam a máquina, antes de a correr.

Evitando “batota” nos exames remotos

Para evitar que isto acontecesse, de os alunos duplicarem as máquinas virtuais para correr os exames, foi gerada a funcionalidade de autenticação *web*, para que só com autorização remota do formador é que pudessem deixar seguir o exame.

Assim, cada aluno, só inicia o exame uma única vez, e caso alguma coisa falhe, só volta a conseguir iniciar se o formador der autorização remota.

Pode implementar-se a regra de que cada autorização nova a um exame desconte 20% da nota, para evitar que alguém remotamente diga que o computador desligou para repetir, que apesar de outras protecções existentes que não mencionarei aqui, alguns podem tentar.

Pode parecer que todos querem fazer batota ao se ler isto, o que não é verdade, mas é importante eu referir porque razão criei este tipo de funcionalidades, pois haver justiça nas notas é sempre essencial.

Para isso dá jeito formador ter um painel de controlo, que será explicado de seguida, sem antes explicarmos que parâmetros usar para criar exames com autenticação remota.

Como forçar nos argumentos de um exame, a Autenticação Remota Via *Web*

Quando preparamos um exame para ser distribuído, como explicámos acima, temos de inserir vários parâmetros que serão forçados no exame (e encriptados), e que serão assim executados de forma forçada quando alguém corre um exame.

Relembro que eles seriam encriptados junto com o exame, pelo que os examinados não conseguiriam alterá-los para falsificar (pelo menos de maneira tão fácil).

O formador teria de ter um servidor para usar este sistema, o que para mim não é problema, por ter os meus próprios servidores, seja de *web*, de *email*, etc.

Ou então, o formador teria de conhecer alguém que tenha um, e que não se importe de lhe dar uma página específica dele, com a sua própria base de dados, coisa facílima de fazer, e com o seu próprio painel de controlo, para ele usar. Repito que é algo facílimo de se fazer.

A esses parâmetros acima, teríamos de adicionar três, para ter autenticação remota, explicados abaixo:

O URL de Autenticação

Aqui colocaremos o endereço do servidor que o *software* terá de contactar, ou seja, o endereço da página no servidor que terá o *software* de contactar para enviar o *username* que vai fazer o exame, ao qual o servidor dará como resposta um "sim" ou "não".

Ele insere-se na linha do *--forceArguments*, com o seguinte parâmetro:
--webAuthUrl **http://www.servidor.com/formacao/autenticar.php**

Apenas o que está acima, onde obviamente em vez de colocarmos o endereço "http://www.servidor.com/formacao/autenticar.php", colocaríamos o nosso.

Obviamente, o formador se quiser usar esta funcionalidade, terá de ter um servidor, e uma página neste caso chamada "autenticar.php", que receberia através do método *POST* certas variáveis enviadas de forma invisível (método *POST*, para quem sabe o básico de *Web Development*), que teria o *username* do examinado, entre outros dados, para verificar na base de dados e teria permissão ou não, e retornar depois um "sim" ou "não" ao programa, que daria início ao exame, ou a mensagem de erro de que não teria permissão para o executar.

O utilizador teria de usar o parâmetro *--webAuthUser*, com o seu *username*, como por exemplo:
./Knowledge\ Tester **exame.txt** ***--password*** **xpto** ***--webAuthUser*** **goncalo**

A linha acima, executaria o exame de nome "exame.txt", com a *password* "xpto", e com o utilizador "goncalo".

Nunca precisaríamos de especificar o argumento do servidor remoto, e outros, porque estariam todos já embebidos e encriptados no próprio exame, desde o momento da sua preparação para distribuição.

É importante referir que a *password* seria só usada pelo *software* para a desencriptação do exame (*Rijndael* 256), e não para enviar via *web*, só o *username* (e alguns dados enviados pelo *software*) é que seriam enviados ao servidor.

O *URL* para envio das notas no fim do exame

Aqui colocaremos o endereço do servidor que o *software* terá de contactar, ou seja, o endereço da página no servidor que terá o *software* de contactar para enviar as notas, bem como hora de término do exame feito, e que serão actualizados na tabela final do formador, que será exibida mais à frente.

Ele insere-se na linha do *forceArguments*, com o seguinte parâmetro:
--webAuthUrl **http://www.servidor.com/grade.php**

A partir do momento em que esta informação está embebida no próprio exame, através do parâmetro *--forceArguments* quando preparamos o exame para ser distribuído, sempre que o exame for executado, e terminado, esta página receberá do *software*, um resumo, ou seja, estatísticas, sobre o exame, no mínimo dos mínimos, a data de término do exame, a hora de término, e a nota com duas casas decimais.

O relatório em si, será tratado por outro parâmetro, mencionado a seguir.

O *URL* para envio dos relatórios no fim do exame

Aqui colocaremos o endereço do servidor que o *software* terá de contactar, ou seja, o endereço da página no servidor que terá o *software* de contactar no momento em que o examinado termina o exame, para enviar o relatório inteiro, ou seja, o mesmo texto que é gravado nos ficheiros de relatório quando se termina um exame, com toda as perguntas erradas e certas, os erros, etc, para o servidor.

O servidor depois terá numa base de dados, numa tabela específica, em cada utilizador e hora de exame, etc, o relatório de cada exame, coisa que fica a cargo de quem cria a página (a *API*), bem como tabelas, etc.

Ele insere-se na linha do *--forceArguments*, com o seguinte parâmetro:
--webAuthUrl **http://www.servidor.com/formacao/report.php**

A partir do momento em que esta informação está embebida no próprio exame, através do parâmetro *--forceArguments* quando preparamos o exame para ser distribuído, sempre que o exame for executado, e terminado, esta página receberá do *software*, um relatório igual ao que é gravado pelo *software* no disco sempre que se termina um exame.

Esta funcionalidade ainda não está totalmente testada, poderá falhar, pelo que é aconselhável pedir sempre aos alunos que enviem o relatório por email no fim dos exames.

É óbvio que os relatórios gerados pelo *software* e gravados no disco, têm uma das suas versões encriptada, de forma a que os alunos não a pudessem adulterar.

Sistemas de detecção de *cheating*

Existem sistemas extra, mas que não colocarei aqui para não mostrar a examinados quais as formas de detecção do que fazem quando fazem batota, e que usarei só para mim. :)

Falha na autenticação por parte dos alunos

Eu faço os exames terem não só autenticação via *web* como também com *password*, para estarem encriptados e gerarem relatórios encriptados.

É comum alunos dizerem que a autenticação falha, e nesses casos basta verificarmos se têm *--webAuthUser* utilizador na linha de execução (onde "utilizador" é o seu utilizador), e se a *password* está correcta, que é algo em que falham muito, pois costuma ser uma *hash* MD5 gerada a partir de outra *hash*, e que faz com que se enganem muito.

Exemplo de *script* a enviar a alunos remotos

Por isso o que faço é gravar um pequeno *script* no *website* *www.pastebin.com*, em que meto algo como:

```
mkdir 2023-09-02-exame-xpto
cd 2023-09-02-exame-xpto
wget www.goncalo.pt/formacao/2023-09-02-exame-xpto.tar.gz
tar xvfz 2023-09-02-exame-xpto.tar.gz
rm 2023-09-02-exame-xpto.tar.gz
chmod 700 KT
./KT 2023-09-02-exame-xpto.kt
echo "Enviem os três relatórios do exame (tudo o que tenha ".report" no nome do ficheiro) para correio@goncalo.pt!"
```

É óbvio que o executável terá dentro o ficheiro do exame, bem como o ficheiro do executável do nosso *software*, para que tudo funcione bem.

Não nos podemos esquecer no *pastebin* de colocar um *expire date*, de 1 dia ou 1 semana, para o *script* não ficar lá para a eternidade, e de o colocar como *unlisted/private*, para não ser acedido por todos.

Depois enviamos o *script* aos alunos, o *link* gerado, claro está, eles abrem-no num *browser* dentro da sua máquina real ou virtual de *Linux* (normalmente quem não tem usa uma máquina virtual ou um *Windows Subsystem for Linux*), copiam o código, abrem um terminal, colam lá o *script*, executam-no pressionando a tecla *Enter*, e já está, o exame inicia-se, e no fim têm a mensagem a dizer-lhes para onde enviar o relatório.

Para enviar, só têm de ir ao *browser* entrar no seu *webmail* pessoal, e enviar o *email*.

Porquê forçar utilizadores iniciados aos *scripts* e não usar só o *Windows*?

Um dia hei-de migrar o *software* para *Windows* também, estive para o fazer em 2023, mas teria de reescrever todo o código relacionado com o terminal de *Linux/*nix*, porque ele obedece a *standards* bem conhecidos, se virem funciona com simples pressionar de teclas, não requer pressionar de *Enter* após cada linha, etc.

Mas o *Windows*, como é hábito, não respeita esse tipo de coisas, e eu investiguei, e o terminal não é grande coisa, não permite fazer *flush* tão bem quanto no *Linux*, a gestão de tudo, etc. E eu nunca iria perder tempo a refazer algo para um terminal de *Windows*, até porque utilizadores de *Windows*, usam-no por norma por não saberem mexer em computadores (todos sabemos que *Windows* é um sistema operativo no geral para pessoas que não têm grandes conhecimentos de computadores, ou que passam a vida a jogar jogos que não existam em *Linux*).

E se essas pessoas não têm grandes conhecimentos, não deveria obrigá-los a mexer em terminais, daí eu pensei: “Vou criar uma *GUI* e a versão para *Windows* só funcionaria com *GUI* (*Graphics User Interface*), enquanto que a do *Linux* funciona tanto com *GUI* ou sem ela (sendo que sem ela até funciona em máquinas virtuais com 20 MB de tamanho quando compactadas, pois já o meti a funcionar em distribuições de *Linux* com 25 MB de disco e 50 MB de memória sem a *GUI*).

Assim dei início a uma libraria minha para a *GUI*, pois gosto de fazer tudo do zero e por isso não iria usar a *Qt* ou outras.

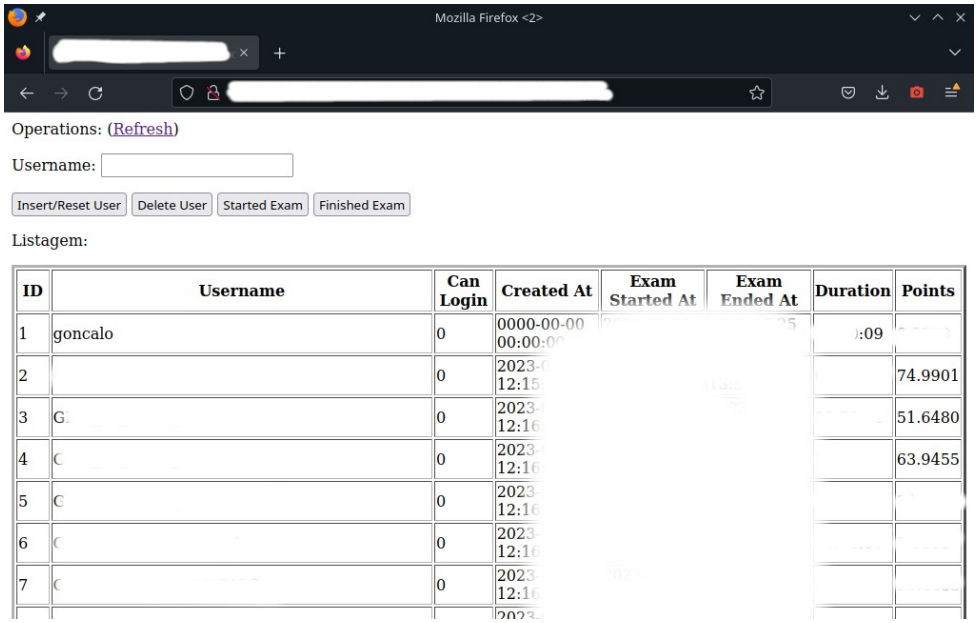
Ao fazê-la, adaptei as partes básicas ao *software*, e assim temos uma versão intermédia, com alguma *GUI* mas ainda sem menus nem outras funcionalidades que terá no Futuro em 2025 quando a retomar, mas é suficiente para migrar para o *Windows*.

Mas acontece que estava farto de mexer nisto e tenho projectos melhores para pegar, como o meu *Game Engine*, os meus *softwares* de Análise Técnica, os meus *Trading Systems*, etc, tudo criado do zero em *C++* também, e por isso este ficará em *stand-by*, e a versão *GUI* para o *Windows* terá de esperar por 2025, até porque pouca gente usa este *software* ainda enquanto formadores e não compensa o esforço.

Assim, o *Windows* quando vir uma versão deste *software*, já será com uma *GUI* bastante mais completa que esta, com menus, opções, etc.

Um Painel de Controlo Remoto para o formador

Abaixo temos um exemplo do primeiro painel de controlo que criei para receber as notas e gerir autorizações, nos meus exames:



Tem tudo o que preciso, o *username*, que pode ser adicionado escrevendo na caixa de texto e carregando em "*Insert/Reset User*", um "*Delete User*", para o caso de termos criado algum por engano, e outros dois.

Um deles, é o "*Started Exam*", para o caso de querermos marcar alguém como tendo começado o exame, e serve para quê? Antes de o exame começar, temos de ter já os *usernames* criados para o exame, e escritos numa lista para dar aos alunos, pois eles remotamente, têm de receber o *link* com o *pastebin* com o *script* que correrão no seu terminal e que irá sacar o *software* e o exame em si.

Cada utilizador escolherá o seu nome na lista, e por norma crio os *usernames* com a data, o módulo, e o nome do utilizador, como por exemplo: "20230902_5116_Goncalo", e será esse o *username* que colocarão após o "*--webAuthUser*" ao executarem o exame.

Isto é bom porque assim podemos ter um painel de controlo com milhares de exames feitos, ordenados de forma descendente pela data, com filtros de pesquisa, etc, e até com a possibilidade de carregar num dos exames, e termos de imediato acesso numa nova página ao seu *report*, com as perguntas todas respondidas, etc.

O "*Started Exam*" existe para quê? Porque se faltarem 3 alunos, temos de clicar lá no "*Started Exam*" com o *username* deles 3, para evitar que os outros possam user esses *usernames* para terem duas máquinas e irem vendo as certas numa para fazer na outra (sempre a pensar nas formas usadas pelos estudantes para fazer batota, são os dias de hoje).

Assim o exame fica marcado como começado e ninguém o pode começar. Também podemos clicar no "*Finished Exam*", que daria ao mesmo, impossibilitaria o aluno de começar o exame com esse *username*, mas esse simula um resultado falso de "27.00", pelo que não deve ser usado.

Posteriormente criei algo mais evoluído, com uma secção que indica certos comportamentos do examinador ao fazer o exame (como saírem da janela, entre outras coisas), e com autenticação (não recomendo usarem sem autenticação a página de gestão frente aos examinados, senão poderiam ver o *URL* da página e acederem à mesma, sem autenticação, e darem permissão a eles mesmos para recomeçar o exame sempre que quisessem.

É importante também verificar a que horas algum examinado começa o exame, pois se alguém começar o exame meia hora após os outros, pode simplesmente ter esperado que outro bom aluno começasse o exame para depois o fazer por ele (não permitido), ou ter visto o tal *URL* e recomeçado o exame, etc, daí convém essa página estar com autenticação.

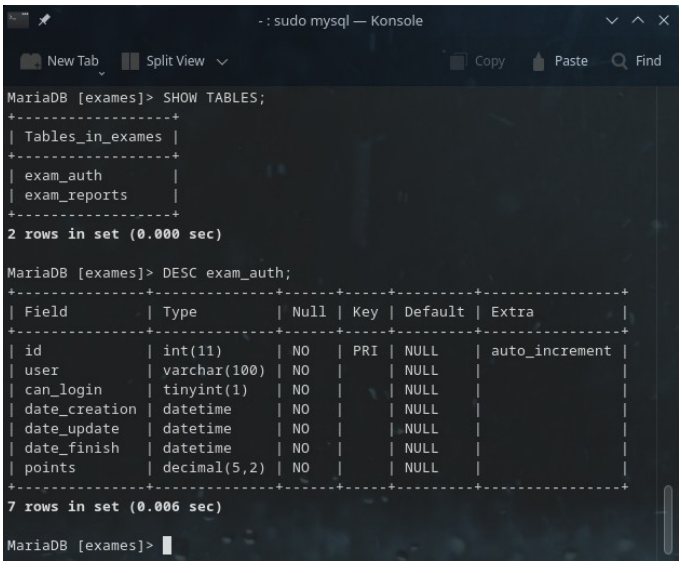
Se alguém ler este manual e quiser usar o *software* com autenticação remota e não tiver servidor, eu poderei (se for conhecido e poucas pessoas) criar uma página improvisada para o sistema funcionar para vocês, mas a da imagem acima e não a minha nova. :)

PS: Foram ambas criadas em *LAMP* (*Linux* + *Apache* + *MySQL* + *PHP*).

Exemplo de uma tabela para guardar dados enviados pelo *software*

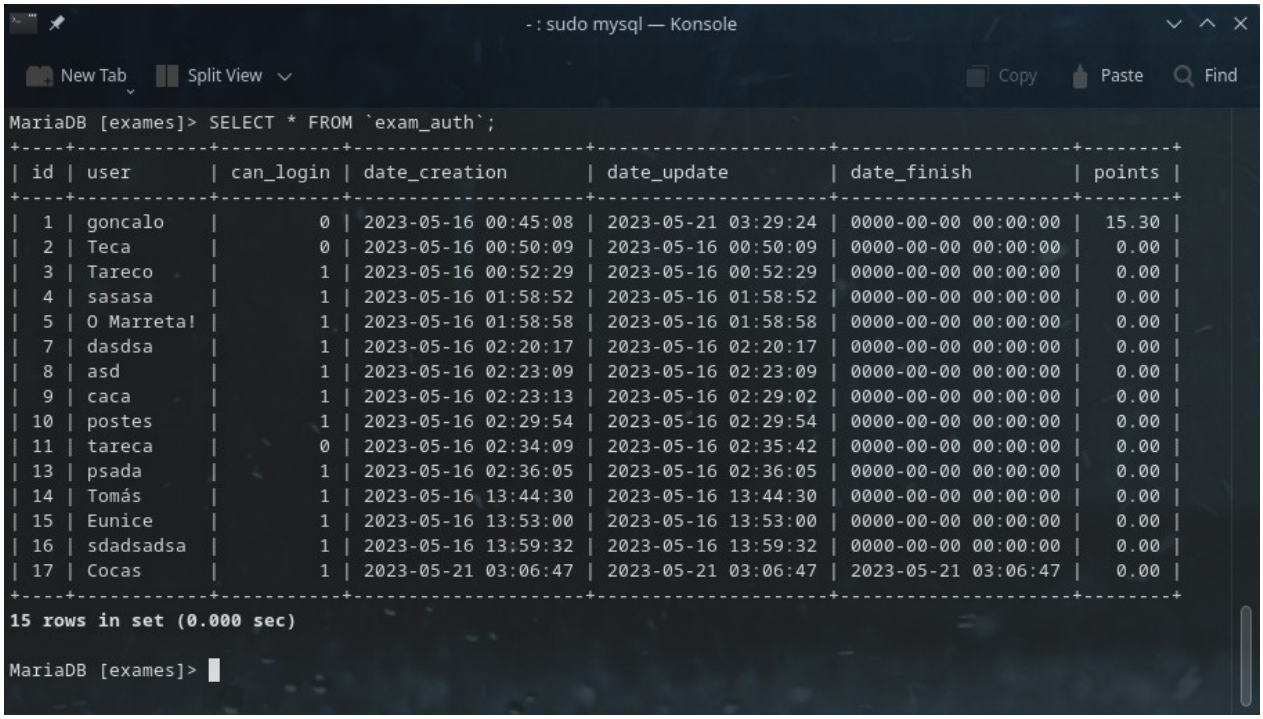
Recordo que esta foi a experiência original, depois criei *API*s mais avançadas para meu uso pessoal, mas esta original funciona e é a que darei aqui para outros simularem.

Para uma tabela em que guardemos os dados descritos no painel de controlo acima, basta-nos algo do género:



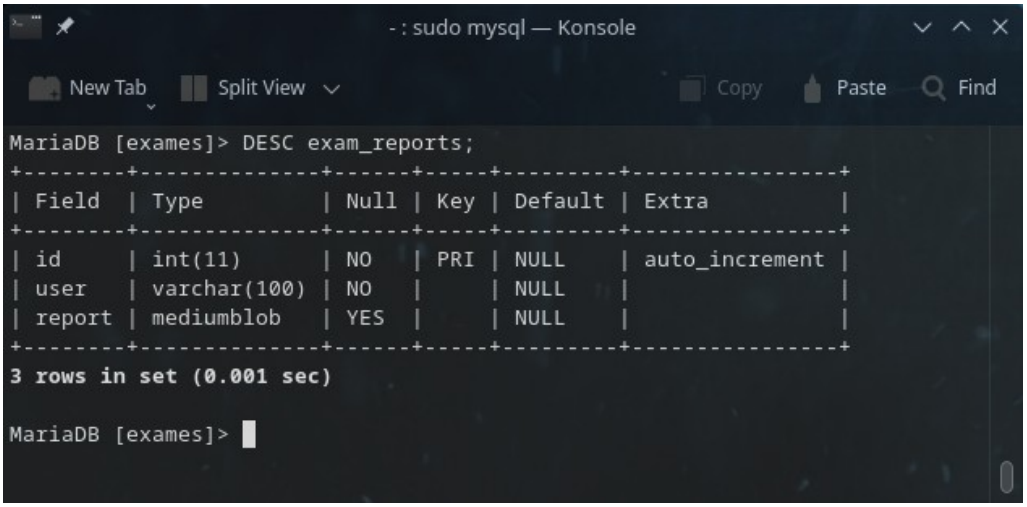
Com estes simples campos que podem ver acima, têm uma tabela *MySQL/MariaDB* onde o *software* pode armazenar dados através da *API*.

O tipo de dados que armazenaria seria:



Como podem ver acima, os que têm "1" no "*can_login*" conseguiriam autenticar-se no exame, ou seja, poderiam fazer o exame, caso esse utilizador fosse usado com um –*webAuthUser*", e temos a data e hora de criação de cada utilizador, o "*date_update*" seria a data de começo do exame (isto era um rascunho original mas funcional), e o "*date_finish*" a data e hora de término, e os pontos no campo "*points*".

Para uma tabela que armazenasse os relatórios, bastaria termos algo como:



Como criar uma mini-*API* para guardar relatórios e fazer autenticação?

A página de autenticação

O que mostrei acima, que foi o rascunho original mas funcional, e bastaria tratarem duas variáveis, a variável *$_POST['u']*, de *username*, e após verificar que na base de dados tinha a possibilidade de seguir exame (o "*can_login*" na tabelas ser "1"), deveria actualizar o campo "date_exam_started" como sendo "*now()*", o "*can_login*" como 0, e ficaria algo assim:

```
$updateQuery="UPDATE `$db`.`exam_auth` SET
`can_login`=0,`date_exam_started`=now() WHERE `user`='$_POST[u]';";
```

E o utilizador começaria o seu exame no *software*, e a base de dados marcá-lo-ia como tendo já feito o exame e já não tendo permissões para o fazer.

A página teria de retornar, um valor específico, que indicaria ao *software* se o utilizador poderia autenticar-se ou não, mas esse valor, obviamente, se algum formador quiser usar, pergunte-me quais os valores, que não convém deixar visíveis aqui nas mãos dos examinados, e quem souber um mínimo de redes ou cyber-segurança compreenderá porquê, pois é algo para estar nas mãos de formadores e não de alunos.

A página que guarda os resultados do exame

O código teria de fazer algo como:

```
$result=$mysqli->query("UPDATE `$db`.`exam_auth` SET
`date_exam_finished`=now(),`points`='$_POST[v]' WHERE `user`='$_POST[u]';");
```

Assim, actualizaríamos a data de fim do exame (a "*date_exam_finished*") com a hora actual, os pontos com a variável "v" recebida por *$_POST*, no utilizador dado pela variável "u" enviada por *$_POST*.

A página que guarda os relatórios do exame

O código teria de fazer algo como:

$updateQuery="UPDATE `$db`.`exam_reports` SET `report`='$_POST[report]' WHERE `user`='$_POST[u]' ORDER BY `id` DESC LIMIT 1;";

Assim, actualizaríamos o campo do relatório, que seria um *blob* de tamanho médio, com o valor recebido através da variável *$_POST['report']*, no utilizador definido pela variável recebida através do valor *$_POST['u']*.

Com pouco trabalho, em menos de meia-hora, e para os experientes 10 minutos, terão 3 páginas que farão tudo o que é preciso para que possam usar essas páginas depois ao criar exames para ter um painel de controlo para gerir exames.

Em caso de dúvidas, se forem formadores, perguntem-me no *LinkedIn*, mas só se for algo que não requira muito tempo da minha parte. :)

É importante referir que o relatório já é enviado escapado (aquele processo conhecido por ser usado para se tentar evitar *SQL Injections*), e não me recordo porque tomei a decisão na altura, mas vai escapado já.

Como preparar um exame para distribuição com Autenticação *Web*

Aqui vamos ter uma repetição do que tivemos acima para tornar um exame pronto a ser distribuído, mas com uma pequena diferença: teremos o mesmo exame preparado para exigir autenticação remota via *web*, através de três endereços, colocados a **negrito**:

./KT --forceArguments exame.txt
--webAuthUrl **http://www.servidor.com/formacao/autenticar.php**
--webGradeUrl **http://www.servidor.com/formacao/grade.php**
--webReportUrl **http://www.servidor.com/formacao/report.php**
--shuffleQuestionsOff 0
--shuffleOptionsOff 0
--writeReport 1
--minVersion 110
--runOnlyOnce 0
--terminal 0
--examMode 1
--retryOff 1
--mentalMode 0
--password "TheAmazingDucksOfApocalypseLetGRSI10DoTheirCExamFinally"

São os argumentos mencionados antes, dos quais relembramos que a versão 110 significa 1.10.

Vejamos as três linhas que importam:

--webAuthUrl **http://www.servidor.com/formacao/autenticar.php**
--webGradeUrl **http://www.servidor.com/formacao/grade.php**
--webReportUrl **http://www.servidor.com/formacao/report.php**

Como sabemos, por já termos falado disto mais acima, a primeira serve para indicar ao *software* para que página pedirá autenticação de um *username* para dar início a um exame, o segundo a página para onde o *software* submeterá as notas e estatísticas globais após o fim de cada exame, e a terceira a página para onde o *software* enviará o relatório final do exame (esta ainda por testar bem e terminar, ficará para 2025).

E se o utilizador inicia com um programa de versão anterior, perde acesso?

Se um utilizador tentar autenticar-se num *software* antigo de versão anterior à requerida pelo exame, ele não perde acesso via *web*, ou seja, não fica "queimado" em termos de tentativas de *login*, pois o *software* obviamente primeiro vê se a versão é a certa, e se não for, é que ele tenta autenticar-se.

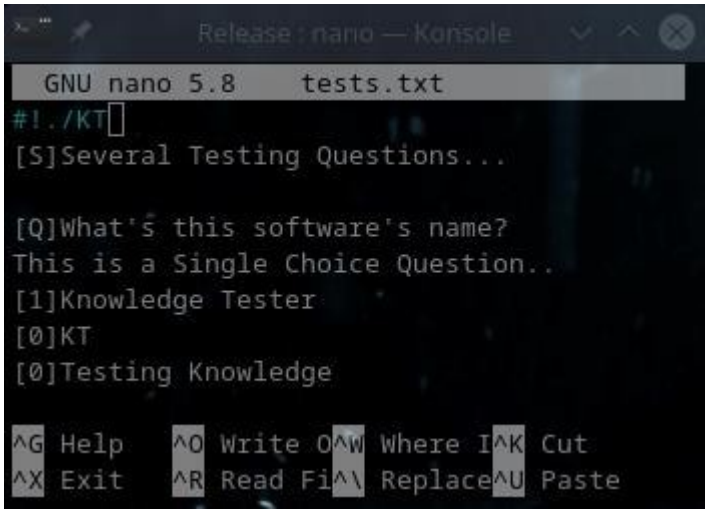
Assim, não há o risco de alguém perder a sua tentativa de fazer o exame, só por ter tentado fazê-lo com um *software* de versão antiga.

Não se coloca o *--webAuthUser* nos *forced arguments*?

Relembro que se colocarmos o *--webAuthUser* nos *forced arguments*, ela seria arrancada dos mesmos, pois não faria sentido, pois ela é usada para autenticar, e não para ficar embebida no exame, senão só permitiria acesso ao tal *username*, e por isso ela é arrancada se for colocada.

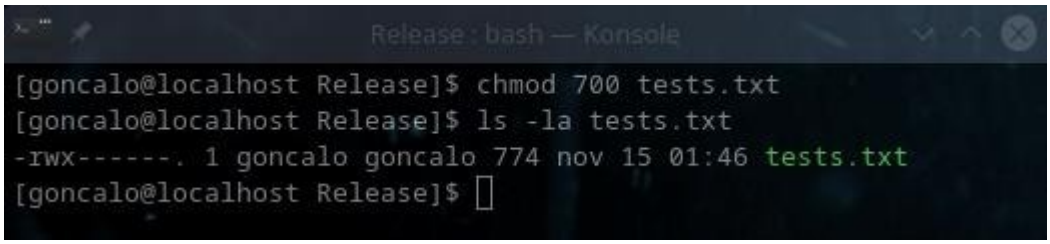
Como transformar um exame num executável

Tal como sucede com *scripts* em *bash* ou *python* ou outros, podemos mandar executar um exame como se fosse um ficheiro executável, se não for encriptado.
Só temos de no começo colocar no lugar usado para definir o interpretador do *script* o caminho para o executável do *Knowledge Tester*, evitando espaços nos nomes:

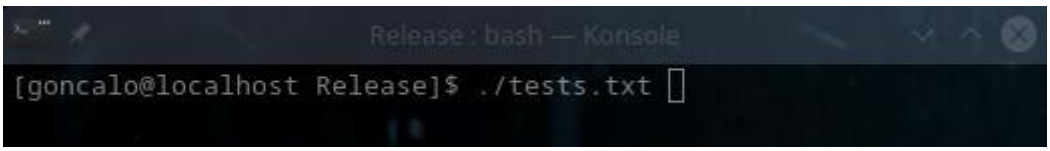


No exemplo acima, definimos o interpretador do exame como sendo o executável de nome "*KT*".

Depois temos de dar permissões de executável ao ficheiro do exame em questão, com um simples "*chmod 700*":



Depois já poderemos executar o comando, desde que haja um *software Knowledge Tester* com o nome "*KT*" na pasta em questão, claro está, e ele correrá como se fosse um ficheiro executável:



É apenas uma ideia que poderá ser útil para facilitar algumas operações, nada mais.

Exportação de exames para outros formatos

Este *software* permite-nos exportar exames (desde que estejam desencriptados ou caso estejam encriptados, que tenhamos a sua *password*, para outros formatos, para poderem ser usados por outras aplicações.

Exemplo de exportação para a plataforma *Kahoot*

A melhor maneira de explicar como exportar algo e mostrando todas as opções, é com um exemplo prático.

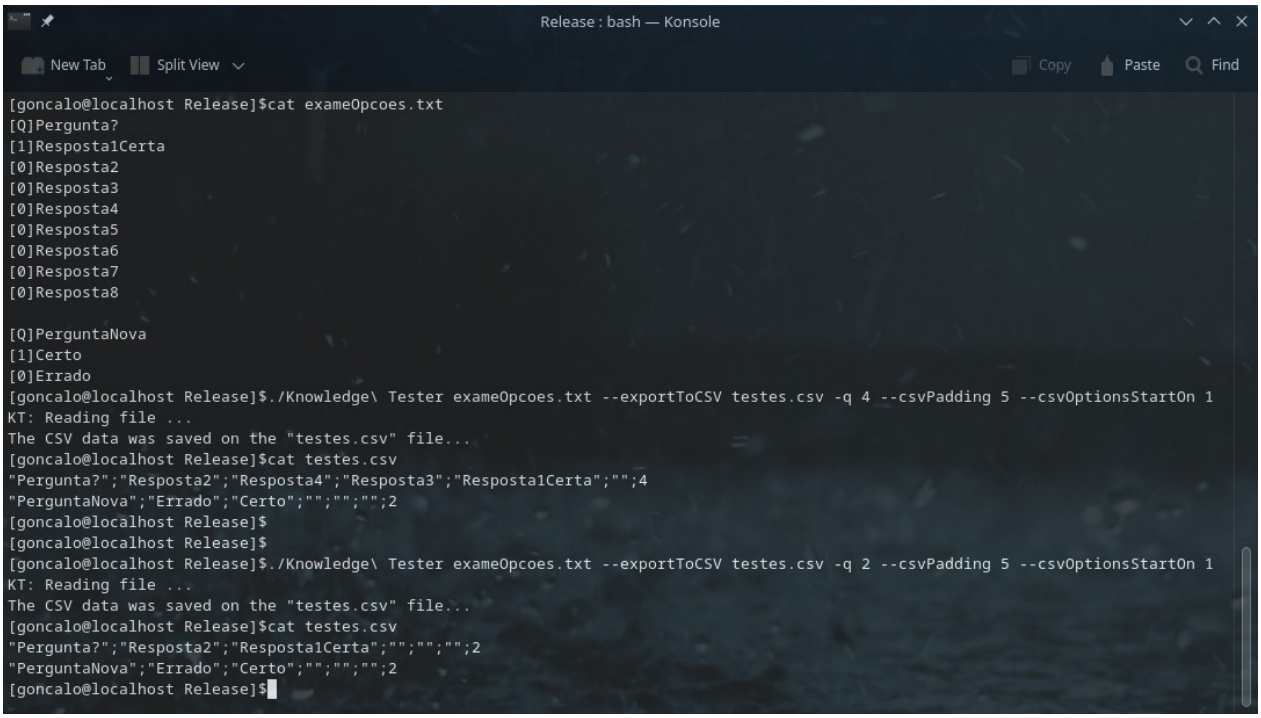
Vejamos uma exportação de um exame, de forma a que possa ser lido na plataforma *Kahoot*…

Com a linha:
./Knowledge\ Tester exameFinal.txt --exportToCSV kahoot.csv -q 4 --csvPadding 5 --csvOptionsStartOn 1

Temos a opção *--exportToCSV* que indica ao *software* que vamos fazer uma exportação para o ficheiro "*kahoot.csv*".

Com o *-q 4* estamos a indicar que exportaremos apenas quatro opções por pergunta. Deveria ser *-o*, mas na altura enganei-me e só corrigirei em 2025.

O programa encarregar-se-á de escolher esse número de opções de forma aleatória, baralha-las, e criar um exame pronto a ser importado no *Kahoot* gravando-o no ficheiro por nós dado:

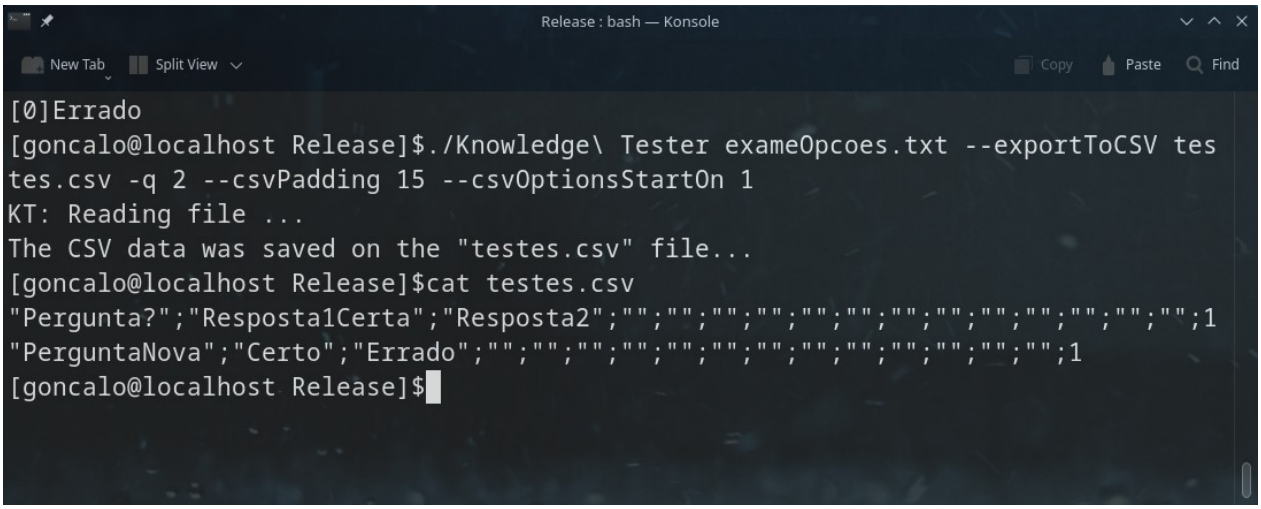


O *software* faz questão de converter todas as aspas das perguntas e respostas, para plicas (de “ para ‘), para evitar que haja problemas na importação.

O outro parâmetro acima usado, o *--csvOptionsStartOn* 1 indica-nos onde começa a sacar as questões para criar o exame.

O programa só exporta se for um ficheiro sem estar protegido, ou seja, um texto *raw*, um ficheiro sem qualquer encriptação, ou então um ficheiro encriptado em que tenham usado o parâmetro *--password* palavra-passe, para se autenticarem, e isto evitará que pessoas que não sejam as autoras do exame, o possam exportar sem pedir a autorização ao criador do mesmo.

O parâmetro "*--csvPadding 15*" diz ao *software* que terá de nos dar 15 possíveis respostas, mesmo que estejam vazias, ou seja, uma espécie de "*padding*", mas em vez de com espaços (passar um "ola" para um " ola", acrescentando os 12 espaços para totalizar 15 de tamanho na palavra), faz um "*padding*" com opções, e neste caso à direita, e isto é para o caso de querermos em certas plataformas importar exames com 15 opções:



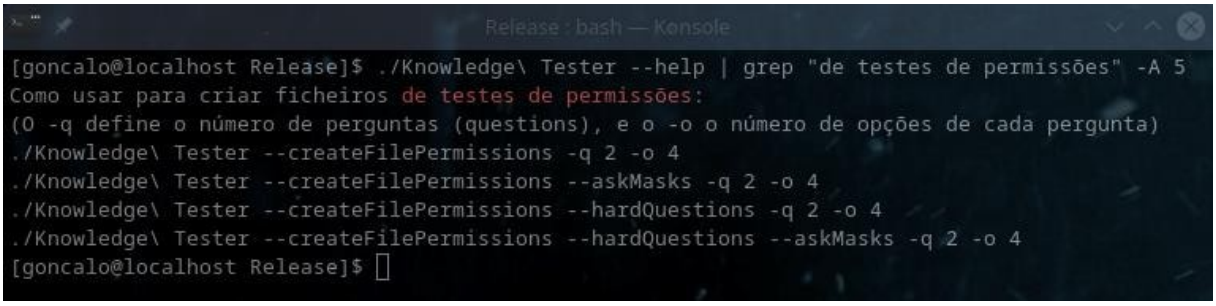
Isto porque no caso do *Kahoot* (e esta funcionalidade nasce por necessidade de eu exportar exames para essa plataforma), ele funciona desta forma, exigindo um número de opções igual ao do exame criado, senão, o último campo com o número da resposta certa, seria confundido com uma das respostas possíveis.

O último campo, como foi dito acima, tem o índice da pergunta certa, e se tem por exemplo "1", é sinal de que o 1º campo logo a seguir à pergunta (ou seja, o 2º), será a resposta certa.

É importante referir que existe um pequeno *bug* que corrigirei só em 2025, que faz com que, se a pergunta ou perguntas no exame tiverem menos de 144 *bytes* de tamanho, receberão como resposta: "*Operation failed: Invalid file name or file doesn't exist…*", porque é o número mínimo que um ficheiro encriptado meu terá neste *software*, e esqueci-me de que podemos usar ficheiros *raw* em texto bruto sem encriptação, mais pequenos que esse número. Se vos acontecer, aumentem o tamanho do ficheiro e já funcionará.

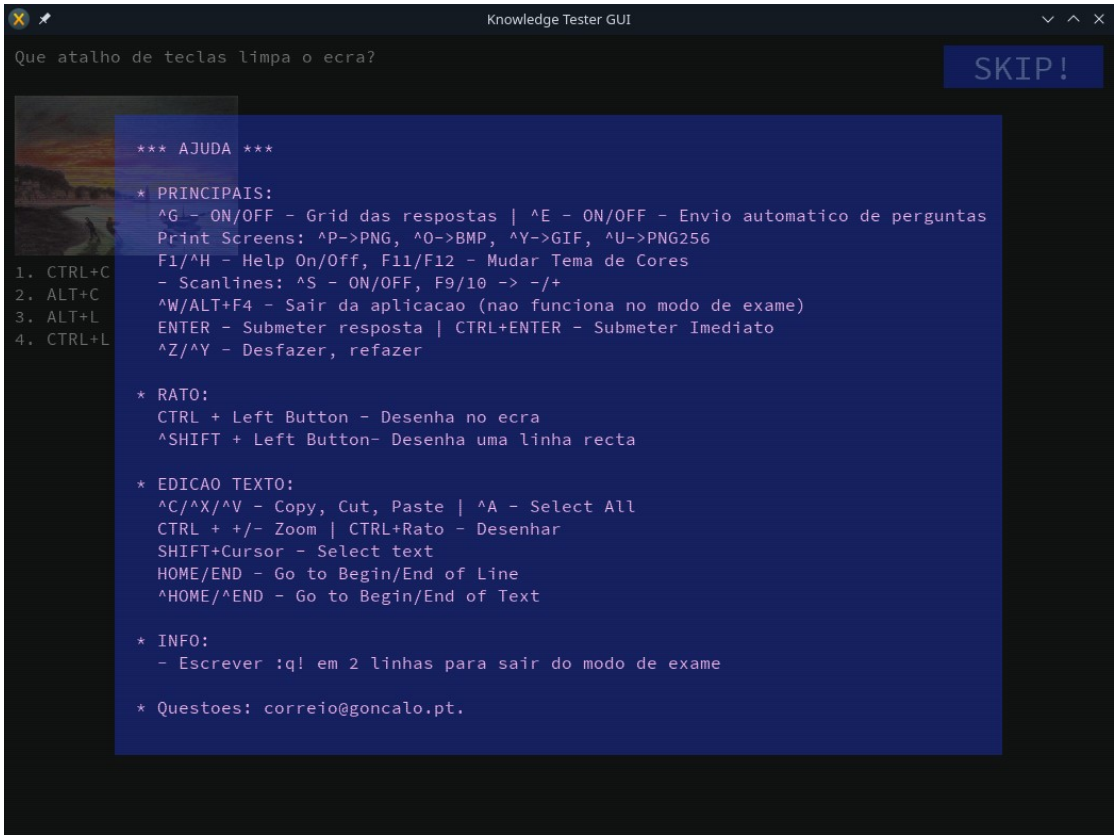
Ajudas e Informações

Se escreverem na linha de comandos "*Knowledge\ Tester –help*", ser-vos-ão mostrados os vários parâmetros e como os usar, apesar de não ser algo tão completo como este manual:



Sem filtros, teremos um texto enorme com muito mais ajudas, claro está, onde bastará fazerem: *./Knowledge\ Tester --help*, e verão toda a ajuda.

Ou em modo gráfico, poderão pressionar a qualquer momento os atalhos *F1*, ou *CTRL+H*, e o ecrã seguinte de ajuda aparecer-vos-á, até que voltem a pressionar um destes mesmos atalhos para que desapareça:



Escusado será dizer que o "^" antes das letras, significa o pressionar da tecla "*CTRL*" antes e durante o pressionar da tecla seguinte.

Sendo assim, um "^*G*" que aparece na Ajuda como sendo o atalho para mostrar ou esconder a *grid* ("rede"), de um editor de texto, usa-se pressionando a tecla *CTRL*, e sem a largar, pressionar a tecla "*G*" depois.

Eu sei que lá aparece o meu endereço de email, mas raramente o consulto, tentem contactar-me pelo *LinkedIn* primeiro se for necessário. :)

Opções Disponíveis para uso no Terminal

Aqui teremos várias opções explicadas, importantes para uso no dia-a-dia. Convém lembrar apenas que poderão não funcionar bem se colocarem as opções antes do nome do ficheiro do exame, ou seja, convém colocarem o nome do exame primeiro, logo a seguir a "*Knowledge\ Tester*", e só depois estas opções.

Repetição de perguntas falhadas em exercícios

Com a opção "*--retryOff*" desligamos uma opção que existe activada por defeito, que é a de repetir toda as perguntas falhadas.

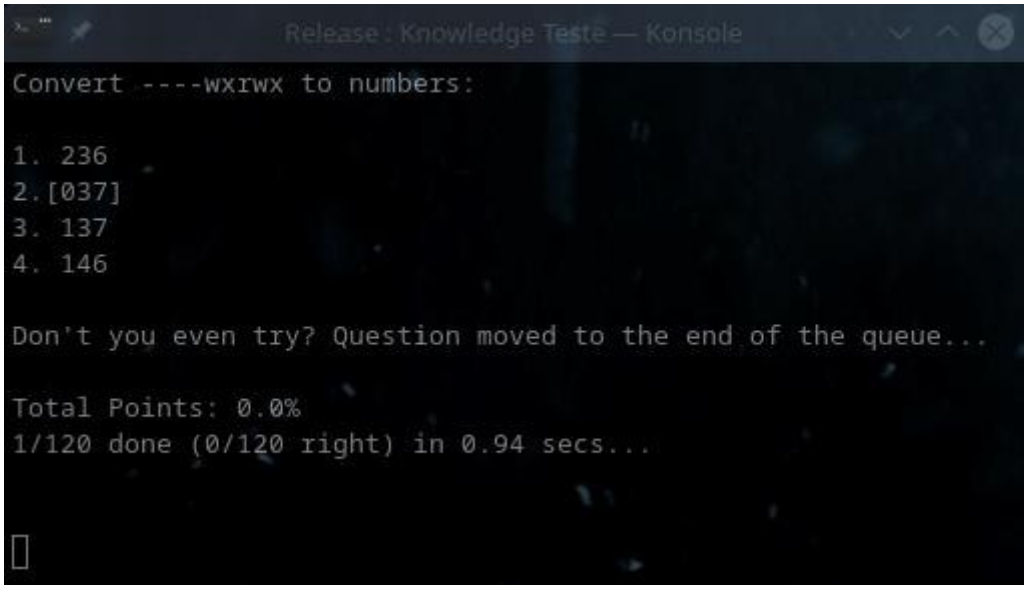
Para facilitar a aprendizagem, e memorização de comandos e matéria, é dada por defeito ao aluno que pratica um exercício, a chance de repetir as perguntas falhadas no fim do mesmo.

Assim, se o aluno falhar a resposta, essa pergunta volta para o fim da lista, e chegando ao fim do número de questões que o exame tem, se existirem questões nesse fim da lista, elas serão apresentadas de novo ao aluno, e se ele continuar a falhar indefinidamente, as perguntas continuarão a lhe ser apresentadas, até que finalmente ele as responda correctamente.

Isto reforça a aprendizagem, porque a pessoa tem de repetir o que não sabe, até memorizar/aprender a matéria, pois o exercício só termina após as perguntas serem todas respondidas correctamente.

Desta forma, se voltar a começar o exercício do zero, ele terá muito mais probabilidades de o terminar sem falhas logo à primeira vez.

Abaixo vemos um exemplo de uma pergunta a ser movida para o final da lista:



Podem ver pela mensagem "*Question moved to the end of the queue…*" acima, que a mensagem foi movida para o fim da lista, sendo que o "*Don't you even try?*" está lá por eu não ter respondido a nada, tendo simplesmente pressionado a tecla *Enter* para seguir em frente.

Se iniciarmos o exercício com o parâmetro "*--retryOff*", elas não serão movidas para o fim do exame, sendo que o utilizador se as falhar ou não responder, não voltará a ser questionado com as mesmas no mesmo exercício, nem sequer no fim da lista.

Repetição de perguntas não respondidas em exames

Durante a execução de exames, também existe esta funcionalidade, mas funciona de forma diferente.

Aqui, não são as perguntas falhadas que passam para o fim do exercício, mas sim as perguntas não respondidas, senão o aluno teria sempre 100% pois responderia a tudo até saber a matéria.

Neste caso, o aluno se responder mal a uma pergunta, é contabilizado com uma nota parcial, ou até 0% se o falhar foi total, e a pergunta não volta a aparecer.

Mas se o aluno pressionar apenas a tecla *Enter*, a pergunta será movida para o fim da lista, e será questionado com a mesma ao chegar ao fim do exame, sendo que o contador de perguntas feitas, não será incrementado, para que só quando responder (nem que seja só com uma letra) a todas as perguntas, é que o contador chegará ao fim e o exame terminará.

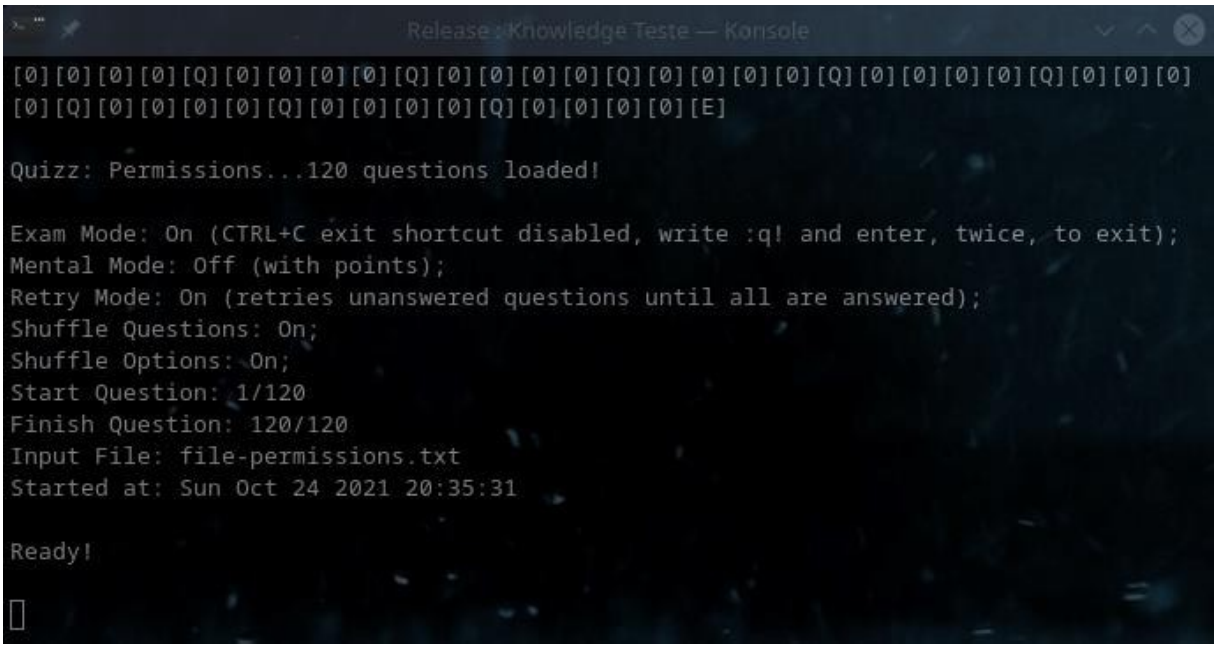
Esta funcionalidade existe para que os alunos nos exames se foquem em responder às perguntas que sabem primeiro, e deixem as que não se lembram bem, para o fim, para não perderem tempo precioso que pode ser gasto a responder às perguntas que sabem.

As perguntas podem ser movidas para o fim da lista as vezes que forem precisas, para que o aluno deixe para o fim só mesmo as que possa não saber mesmo nada.

Neste caso, também o parâmetro "*--retryOff*" funciona, e se for usado, e se um aluno pressionar a tecla *Enter* numa pergunta de um exame, ela não seria movida para o fim da lista, mas simplesmente contabilizada como falhada e com 0% de nota.

O aluno examinado pode saber se a opção em causa está ligada ou não, para saber se pode pressionar a tecla *Enter* para deixar perguntas difíceis para o fim, ou não, no ecrã inicial de arranque do exame.

Por isso há que estar atento ao ecrã de arranque do exame, e verificar se a opção "*Retry Mode*" está activa ou não, e o que diz:



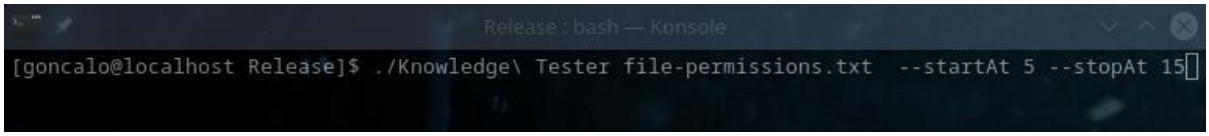
E como podem ver acima, após o "*Retry Mode*", temos escrito "*On*" e "*(retries unanswered questions until all are answered)*", ou seja, o aluno poderá mover perguntas não respondidas para o fim da lista sem problemas.

Correr um intervalo de perguntas apenas

É importantíssimo saber que é possível correr apenas um intervalo de questões num dado ficheiro de perguntas.

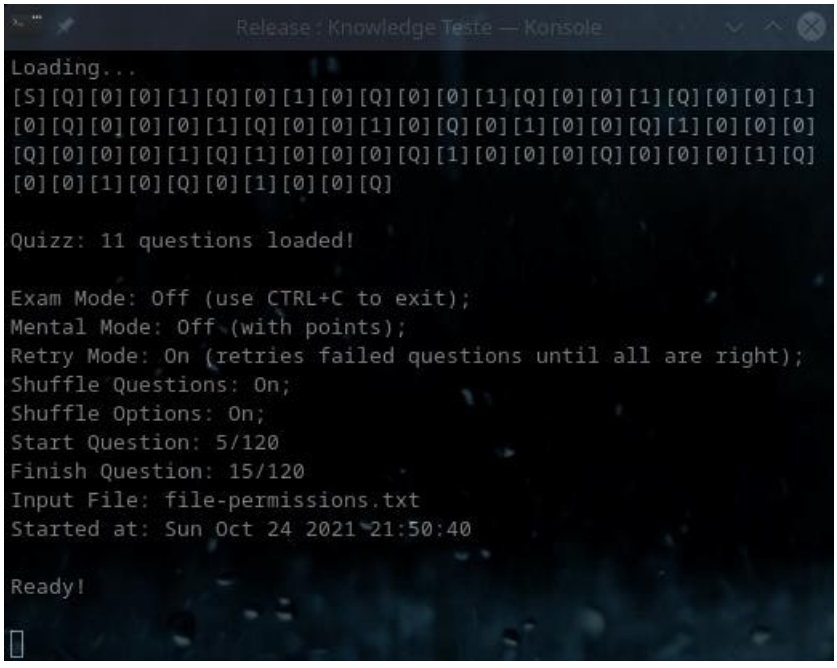
Assim, se tivermos um ficheiro de exercícios com umas 100 perguntas, não será fácil aprender as respostas, mesmo que as falhadas sejam enviadas para o fim, com 100 perguntas, pois se falharmos a 3ª e virmos a resposta correcta, não vamos recordar a mesma tão bem, 97 perguntas depois.

Nesses casos, quando os ficheiros de perguntas são grandes, o formando ou aluno, pode usar as opções "*--startAt*" e "*--stopAt*" para definir em que pergunta começa e acaba o exercício.
No exemplo abaixo, definimos que num exame de 120 perguntas, vamos executar apenas as perguntas entre a quinta e décima quinta perguntas:



Na opção acima, com o "*--startAt 5*" e o "*--stopAt 15*", vamos correr um exame onde só responderemos às questões entre a quinta e a décima quinta, ignorando todas as outras, e apesar de serem 120 perguntas, como só vamos responder a 11, cada uma terá um valor de 9,09% de nota, obviamente.

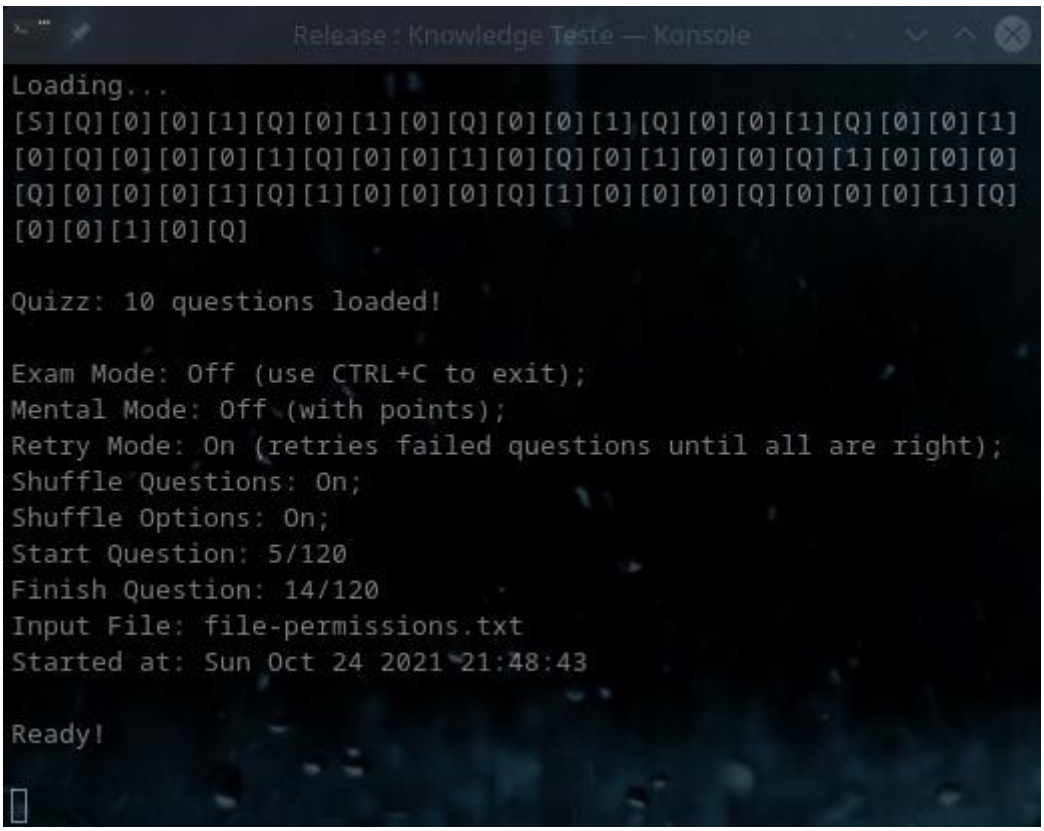
E porquê 11 perguntas? São 11 porque se definimos começar em 5 e acabar na pergunta 15, ele vai correr as 10 entre 5 e 14, e a 15ª também. Ou seja, as perguntas definidas no argumento do parâmetro "*--startAt*", e as definidas no argumento do parâmetro "*--stopAt*", são todas contabilizadas:



Assim, se desejarmos apenas 10 perguntas, entre a 5ª e a 15ª, excluíndo a 15ª, ou seja, na realidade, responder às perguntas 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, mas ignorando a 15ª, teríamos de correr o comando com o "*--startAt*" em 5 e o "*--stopAt*" em 14:

Pelo exemplo acima, já teríamos 10 perguntas, de 4 a 14, e valendo cada uma 10% de nota, por serem dez no tota, e isto pode parecer confuso à partida, mas não é, pois tem mais sentido dizer para se correr da pergunta 1 à 10, 11 à 20, 21 à 30, 31 à 40, do que dizer que arrancamos com 1 à 11, 11 à 21, 21 à 31, etc.

Faz mais sentido assim, incluir ambas as perguntas, a do começo e a do fim. Vejamos o resultado do comando acima:



E assim conseguem-se filtrar algumas perguntas num exame com muitas perguntas, e dividindo-o assim em várias secções, para serem memorizadas facilmente.

É importante referir que mesmo que sejam baralhadas as perguntas, só são lidas no exemplo acima as perguntas da 4ª à 14ª, e só depois de lidas é que são baralhadas, pelo que é seguro filtrar perguntas assim.

Baralhar perguntas e opções

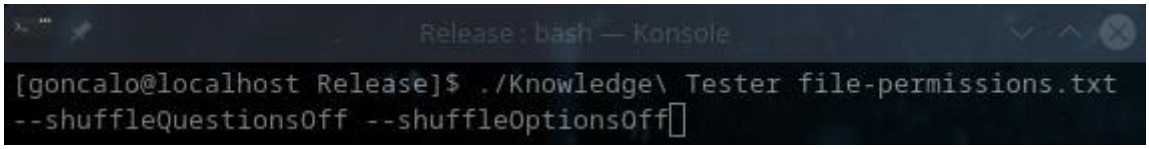
Existem dois parâmetros que nos permitem desligar o que está activo por defeito, que é o baralhar das perguntas e opções das perguntas.

Com o parâmetro "*--shuffleQuestionsOff*" podemos desligar o baralhar das perguntas, e elas aparecerão na ordem em que estão escritas no ficheiro de perguntas ou exame.

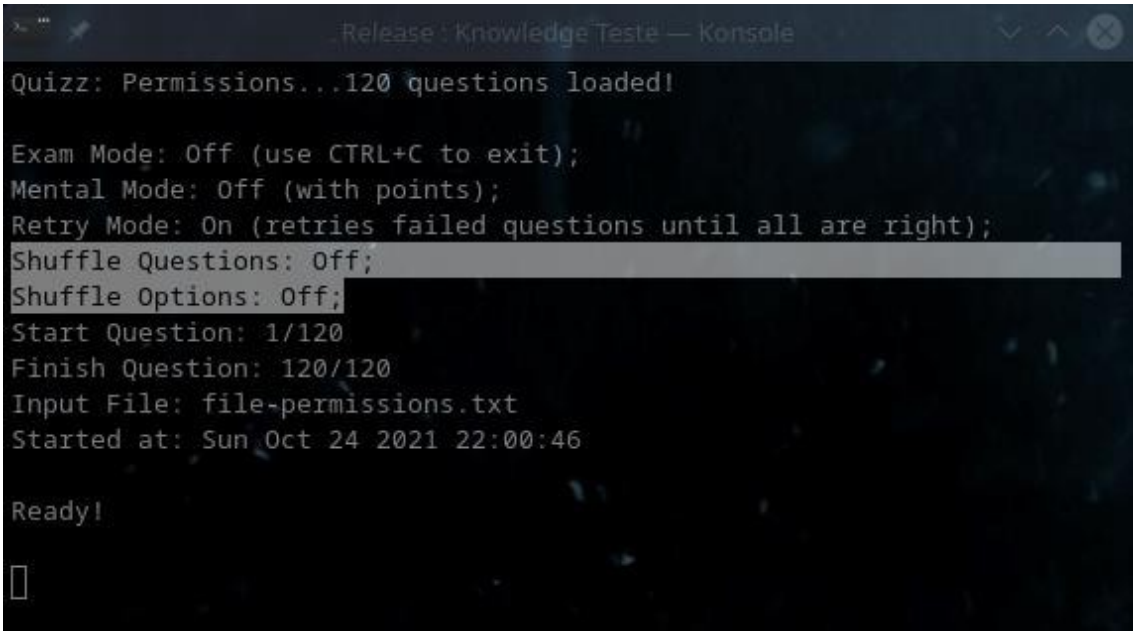
Da mesma maneira, com o parâmetro "*--shuffleOptionsOff*", podemos desligar o baralhar das opções de cada pergunta, pelo que surgirão pela mesma ordem em que estão escritas no ficheiro de perguntas ou exame.

No caso das opções não há muito sentido, mas o desligar o baralhar de perguntas pode ser útil em casos de laboratórios, em que simulamos os passos necessários para uma operação com muitos comandos, em que dedicamos uma pergunta e explicação a cada um dos passos, e em que nesses casos necessitamos de ter perguntas a sair pela ordem em que estão nos ficheiros.

Abaixo fica um exemplo de como se corre um exame com ambas as opções desligadas:



E depois pelo ecrã de arranque veremos que já estão desligados ambos os baralhar de perguntas:



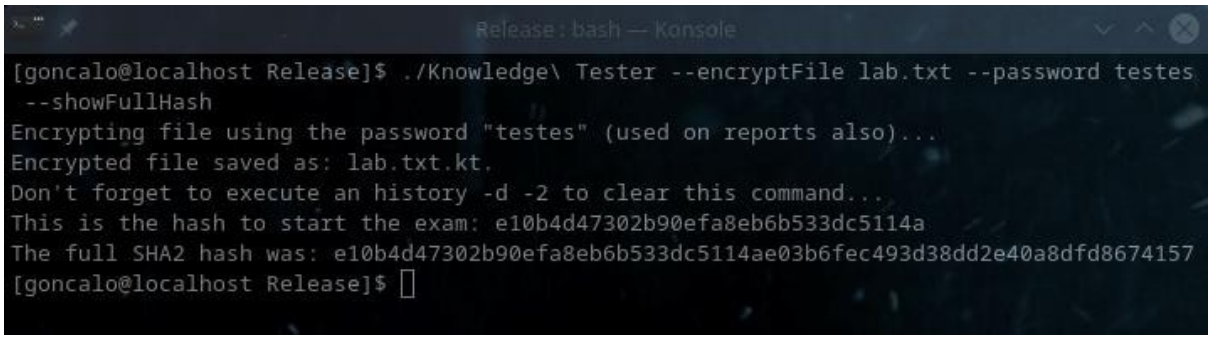
E assim se desligam as funcionalidades de baralhar tanto as perguntas como opções, que podem ser obviamente ambas chamadas individualmente, pois nenhuma delas necessita da outra.

Opções de encriptação de ficheiros

Aqui são apenas referidas algumas poucas funcionalidades, como a de mostrar a *hash* inteira aquando da encriptação de um ficheiro de exame, que por norma ela é encurtada, para facilitar a sua transmissão aos alunos, e por sabermos que mesmo assim não deixa de ser mais do que segura.

A opção *–showFullHash*

Com o parâmetro "*--showFullHash*", surge-nos a linha no fim, que nos diz "*The full SHA2 hash was:*" seguida de uma *hash* inteira:

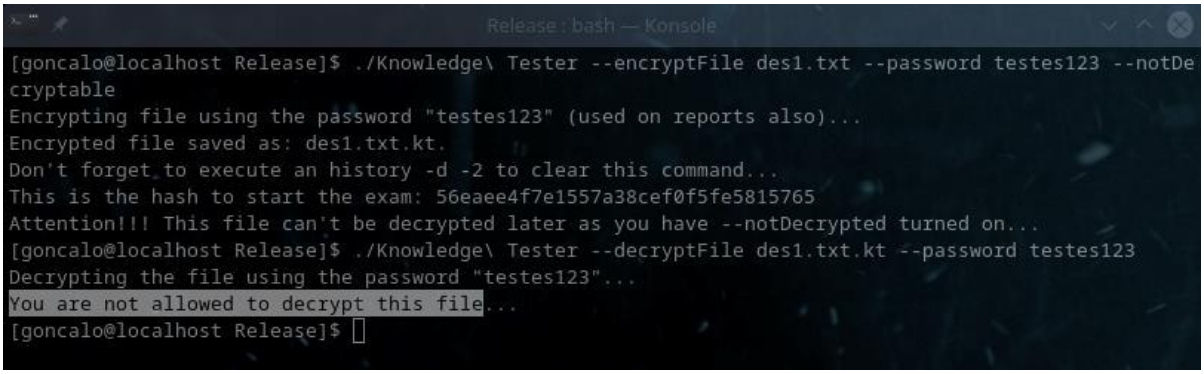


Mas é importante referir que nos basta a *hash* encurtada para abrir exames, tudo o resto é descartado pela aplicação, pelo que este parâmetro só é mencionado aqui para efeitos de teste por parte dos formadores ou outras pessoas que queiram criar exames.

A opção *–notDecryptable*

Com esta opção activada durante uma encriptação de um ficheiro, o mesmo não voltará a poder ser desencriptado, por isso há que ter cuidado com ela, pois se perdemos o ficheiro original, e o encriptado tiver sido criado com esta opção, nunca mais recuperaremos o ficheiro de texto original, pelo que teremos de o recriar do zero se o quisermos alterar.

Abaixo podemos ver como a usar, e como é impossível depois desencriptar o ficheiro mesmo possuindo a *password* original:



O parâmetro de versão mínima ("*--minVersion*")

Este parâmetro pode ser definido em conjunto com o "*--forceArguments*", e deixa-nos registar no exame a informação de que o exame em questão só pode correr num *software Knowledge Tester* com uma versão igual ou acima da definida por este parâmetro.
Essa versão tem de ser dividida por 100.

Se usarmos o parâmetro "*--minVersion 19*", significa que o software só correrá se a versão do *software* em si for *0,19* ou superior, e uma "*99*" significaria versão *0,99*, uma "*1*" significaria versão 1,00, etc.

Isto permite-nos garantir que o exame é corrido numa aplicação que tenha todas as funcionalidades requeridas pelo exame em questão, para evitar que os resultados do mesmo possam ser manipulados, ou o exame executado de maneira não inicialmente pensada.
Para isso só temos de colocar esta linha à frente da linha de comandos que corramos com a opção "*--forceArguments*", que deverão procurar no índice deste manual, e ela será guardada no exame.

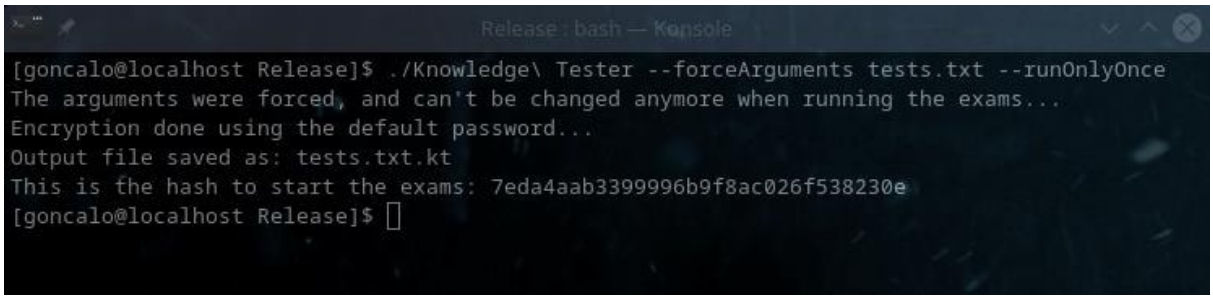
É de notar, que só é guardada em exames encriptados, pelo que se desencriptarmos o exame, ela deixará de estar activa (ver mais nas secções sobre como encriptar exames).

Proteger exames para só executarem uma vez ("*--runOnlyOnce*")

Com o parâmetro "*--runOnlyOnce*" inserido com o "*--forceArguments*", o exame passa a estar protegido e só deixa que seja corrido uma única vez, e quem tentar correr novamente verá a sua tentativa frustrada.

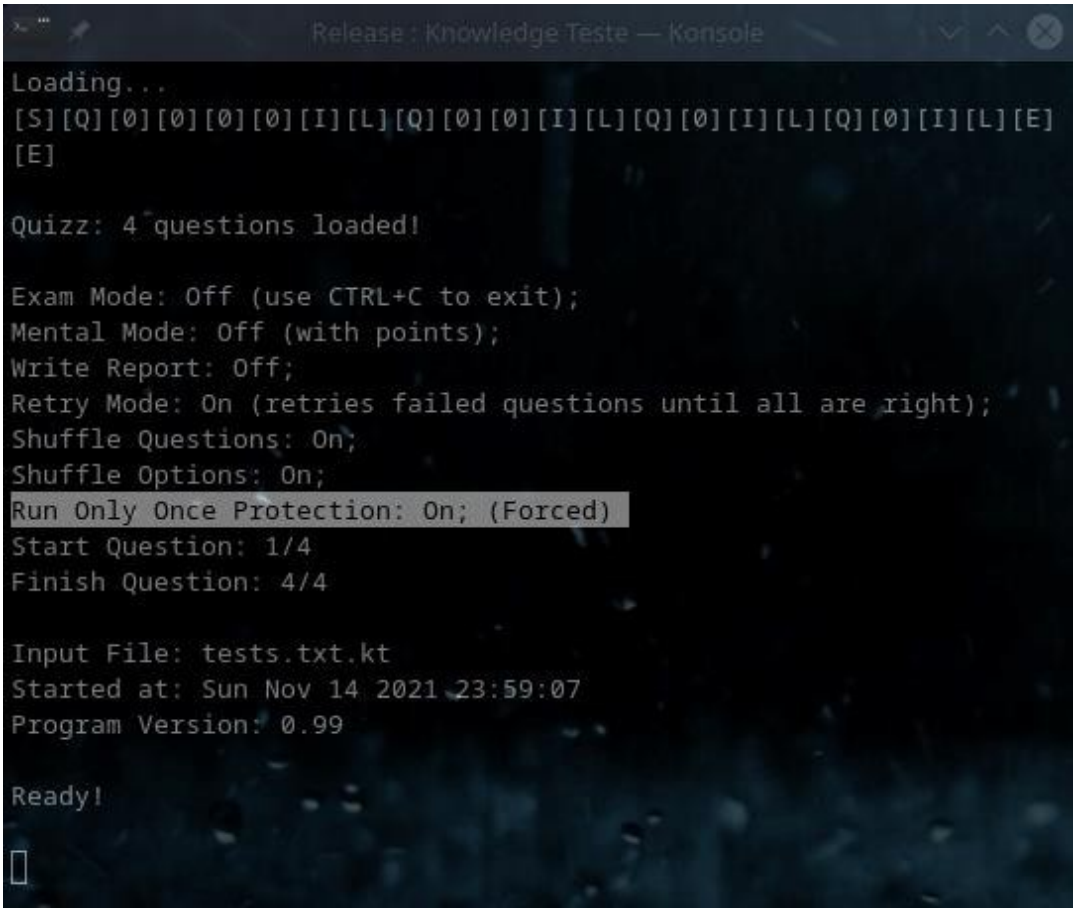
Esta protecção existe para evitar que alunos cancelem o seu exame assim que começa a correr mal e o recomecem, e com ela, assim que o exame é iniciado, não pode voltar a ser reiniciado.

Quando usada junto com o argumento "*--forceArguments*", ela é guardada no ficheiro na sua forma encriptada, e só desaparece se usarmos a opção contrária do "*--removeRunOnlyOnceProtections*".



Quando forçamos argumentos, como o ficheiro é encriptado por defeito, é-nos dada uma *password* para que possa ser aberto, e por isso se queremos que ele veja as opções forçadas com uma determinada *password* que não a por defeito, devemos acrescentar o argumento "*--password* palavra-passe" à linha de comandos.

Mas se tentarmos executar o comando, veremos que já tem forçada a opção de correr apenas uma vez, aparecendo "*(forced)*", à frente da mesma:



Se tentarmos correr o mesmo exame novamente, teremos nada mais do que uma mensagem de erro dizendo que tal já não é possível porque o exame já correu uma vez:

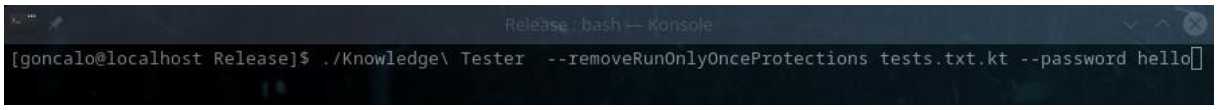
E é de notar que nem que se troque o exame por um exactamente igual, se consegue executar tal exame novamente, é uma protecção básica e simples, mas minimamente eficaz, que é o que existe de momento nesta fase de desenvolvimento do programa.

Remover protecções de correr apenas uma vez ("*--runOnlyOnce*")

Com o argumento "*--removeRunOnlyOnceProtections*", usado em conjugação com o argumento "*--forceArguments*", o ficheiro encriptado é aberto, essa sua opção é removida, e é guardado já sem essa protecção.

Este argumento existe para o caso de alguém querer remover essa protecção a um exame, para que os alunos o possam correr as vezes que forem necessárias.

Para isso temos de invocar esse argumento "*--removeReadOnlyProtections*" seguido do nome do ficheiro em questão, sem esquecer de referir a *password* original usada para encriptar/proteger o ficheiro, e tem de ser essa original, senão qualquer aluno/formando seria capaz de o fazer com a *password* de abertura dos exames:



Após remover as protecções, teremos a seguinte mensagem, e poderemos correr novamente o exame agora desprotegido:

Geradores de Exercícios Embebidos

Existem vários exercícios já pré-embutidos no *software*, como um de mover pastas, um de permissões de ficheiros, e um de mover pastas e mudar nomes.

São exercícios que compensa existirem em separado, pois fazem o aluno ou formando praticar em poucos minutos algo que levaria horas de uso real do sistema a praticar, em termos de experiência, pois no dia-a-dia é raro alguém estar a mudar constantemente permissões de ficheiros, e com este exercício acabam por o fazer centenas de vezes em dez minutos.

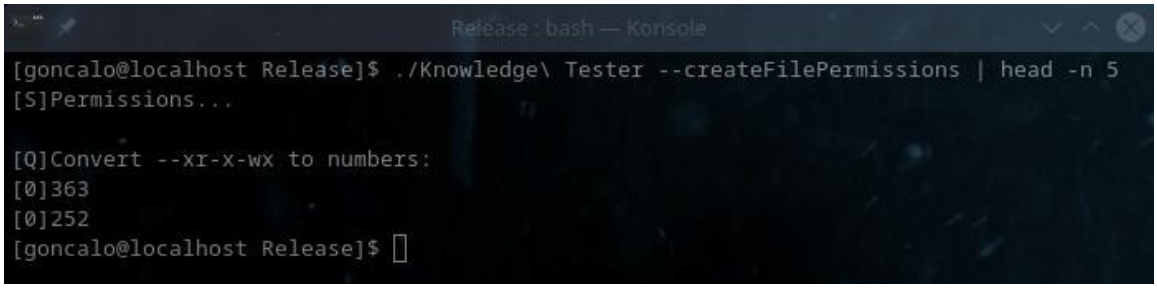
Este tipo de exercícios compensa ter neste tipo de *software*, e é provável que outros sejam adicionados em futuras versões deste *software* gratuito.

Exercícios de Permissões de Ficheiros

Como criar o exercício

O exercício é gerado através de uma linha de comandos como tudo o resto, com o comando e parâmetro "*Knowledge\ Tester --createFilePermissions*", e tem quatro estilos possíveis de exame, conseguidos com a combinação de duas opções, a "*--askMasks*", e a "*-- hardQuestions*", que conjugadas nos dão quatro tipos de exercícios possíveis.

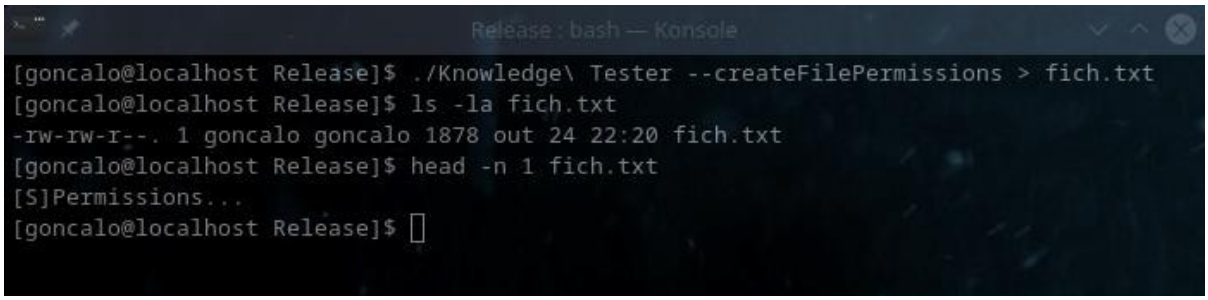
Abaixo temos o invocar do comando que nos dá as questões:



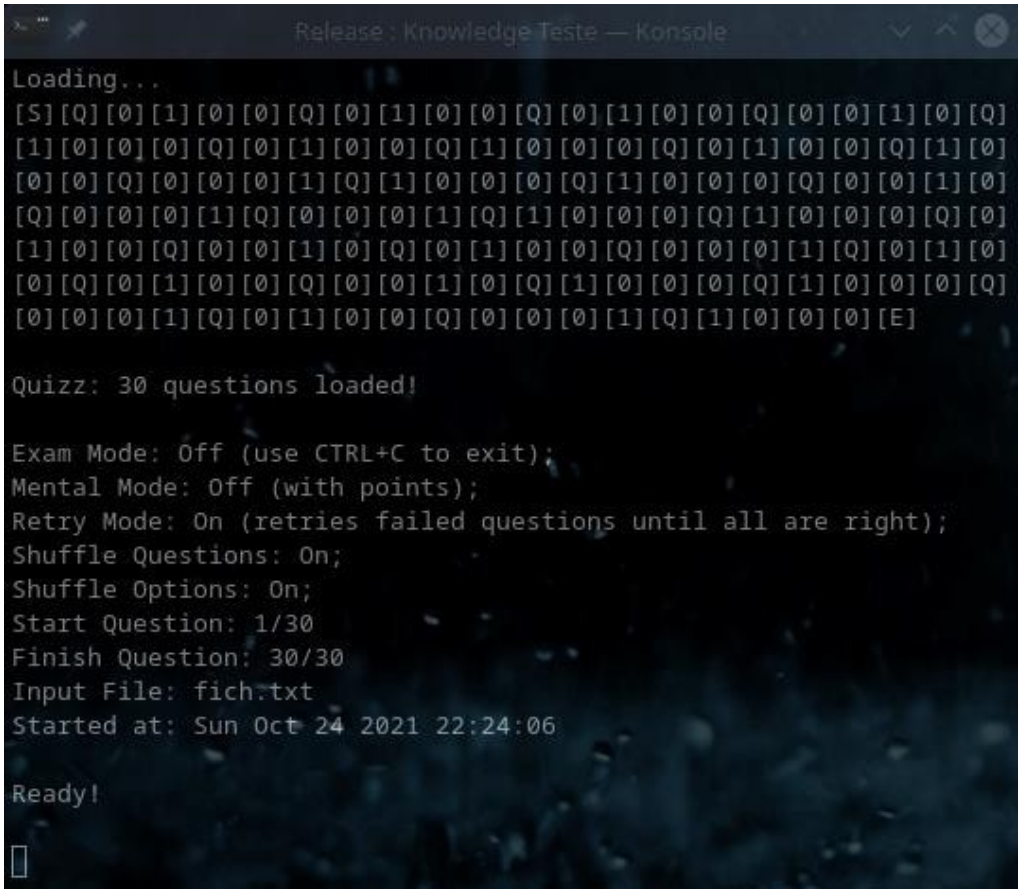
Mas se como podemos ver acima (filtrado o *output* com o comando "*head -n 5*" para mostrar apenas as primeiras 5 linhas do exercício), o *software* envia-nos as perguntas para o ecrã.

Sempre que repetirmos este comando, poderemos ver que nos cria questões diferentes, "aleatoriamente".

Para fazermos exercícios com este resultado, temos de acrescentar um "> *ficheiro.txt*" ao fim da linha, para gerar o ficheiro com respostas:



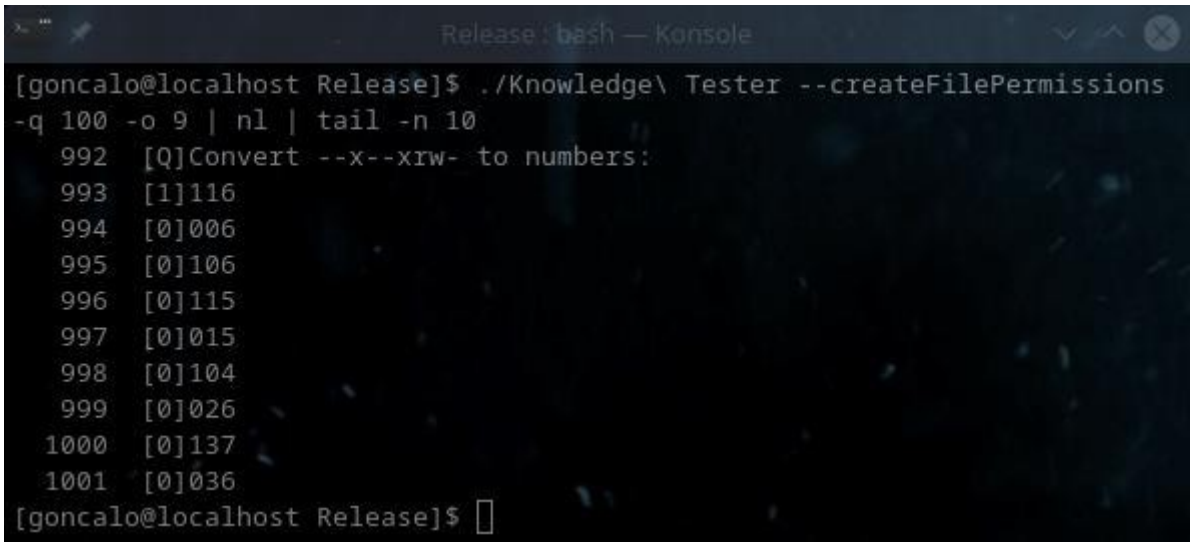
Na imagem acima, vemos que o ficheiro de nome "*fich.txt*" foi assim gerado, e que contém neste momento as perguntas para serem respondidas, que são por defeito 30:



Como definir o número de questões e opções por exercício

Como podemos alterar o número de questões a ser apresentada por defeito?

Se executarmos "*Knowledge\ Tester --help*" podemos ver que temos os parâmetros "*-q*" e o "*-o*", sendo que o "*-q*" nos permite definir o número de questões que vamos ter no exercício, e o "*-o*" nos define o número de opções por cada questão do exercício:



No exemplo acima podemos ver que definimos 9 opções por questão, e 100 questões (das quais só vemos uma no ecrã graças ao comando "*tail -n 10*"), e podemos ver que foram 100 questões, dado que com o comando "*nl*" (que nos coloca o número da linha antes de cada linha), podemos ver que são perto de 1000 linhas, que correspondem a 100 questões onde cada questão tem uma linha para a pergunta e nove para as opções.

É importante referir que o número mínimo de opções é 2, sendo que se for definido um número inferior, teremos 2 opções por questão.

O número mínimo de questões é 1, sendo que se definirmos 0 questões, teremos uma.

E o número máximo de opções é 9, sendo que se definirmos mais do que 9, teremos 9, e isto é importante, dado que só podemos responder usando as teclas de 1 a 9, e se tivéssemos 20 opções e a certa fosse a opção 15, não conseguiríamos responder com o premir de uma tecla, daí são limitadas a 9, para serem respondidas com as teclas de 1 a 9.

Opções disponíveis

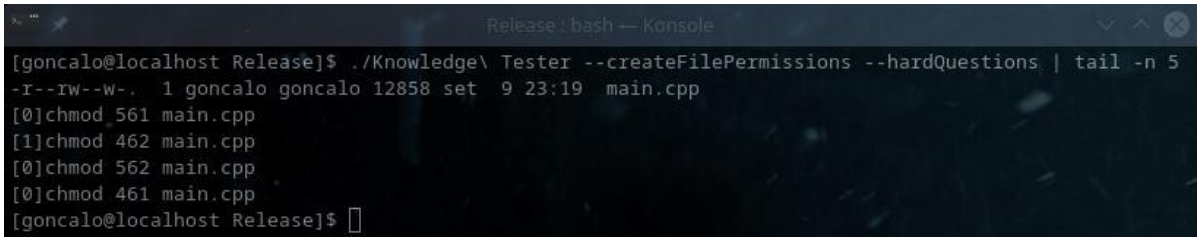
O parâmetro --askMasks

Este parâmetro neste exercício faz com que sejam pedidas ao utilizador as máscaras "*rwxrwxrwx*" sendo dados os números (ao invés do contrário):

Este tipo de perguntas em conjugação com as anteriores enriquecem o exercício, e será explicado mais abaixo como as juntar num único.

O parâmetro –hardQuestions

Com este argumento, o "*--hardQuestions*", tudo é ligeiramente dificultado, pois as máscaras "*rwxrwxrwx*" passam a ser iguais ao que se vê numa *shell* tradicional de *Linux*, tendo um "-" antes de cada linha (a representar o ficheiro), e podemos ver como fica aqui no método tradicional sem "*--askMasks*" mas com *"--hardQuestions*":



Podemos ver acima que a pergunta já fica mais realista, e algo mais difícil, daí a *tag* ter como nome "*--hardQuestions*", apesar de que na realidade foi também por falta de imaginação da minha parte para um nome melhor.

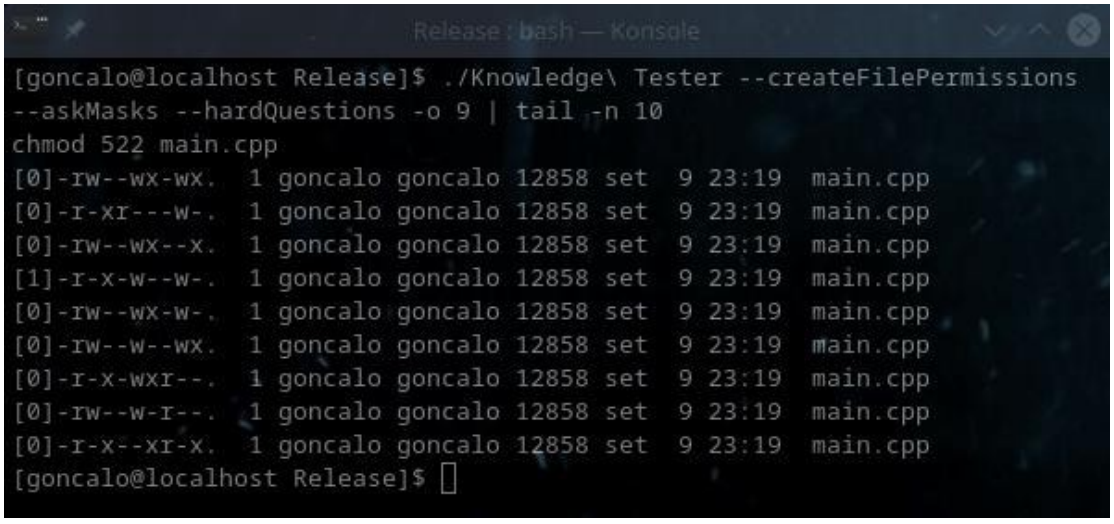
Por norma o ideal é sempre ser usada a *tag* "*--hardQuestions*", pois é mais realista, e o objectivo é sempre o aluno/formando estar preparado o mais possível para a realidade, para distribuições reais de *Linux* com as suas *shells* reais.

Vejamos como fica a mesma *tag* "*--hardQuestions*" conjugada com a *tag* "*--askMasks*":

Bastante mais realista, e por norma basta que o formando ignore o "-" inicial, que é usado para no *Linux* representar um ficheiro (sendo que "*d*" é para directorias, "*l*" para *links*, etc), e tudo se torna mais fácil.

Assim, o ideal é conjugar exercícios com "*--askMasks*", com exercícios com "*--hardQuestions*", e outros com a segunda mas sem a primeira, e veremos mais abaixo como os criar.

Agora para dificultar mesmo a tarefa, basta juntarmos o parâmetro "*-o 9*" para aumentar o número de opções, e ficaremos com:



Até dá vontade de tentar, não?

Na realidade, apesar de esta pergunta ser uma brincadeira considerada por alguns de mau gosto, isto torna-se bem mais fácil do que parece, com alguma prática, pelo que o ideal é a pessoa praticar sempre com o número máximo de opções (9) e com as *tags* "*--hardQuestions*", tenham a "*--askMasks*" activa ou não.

Juntar vários exercícios num só

Agora, como juntar de forma fácil os quatro tipos de exercícios num só?

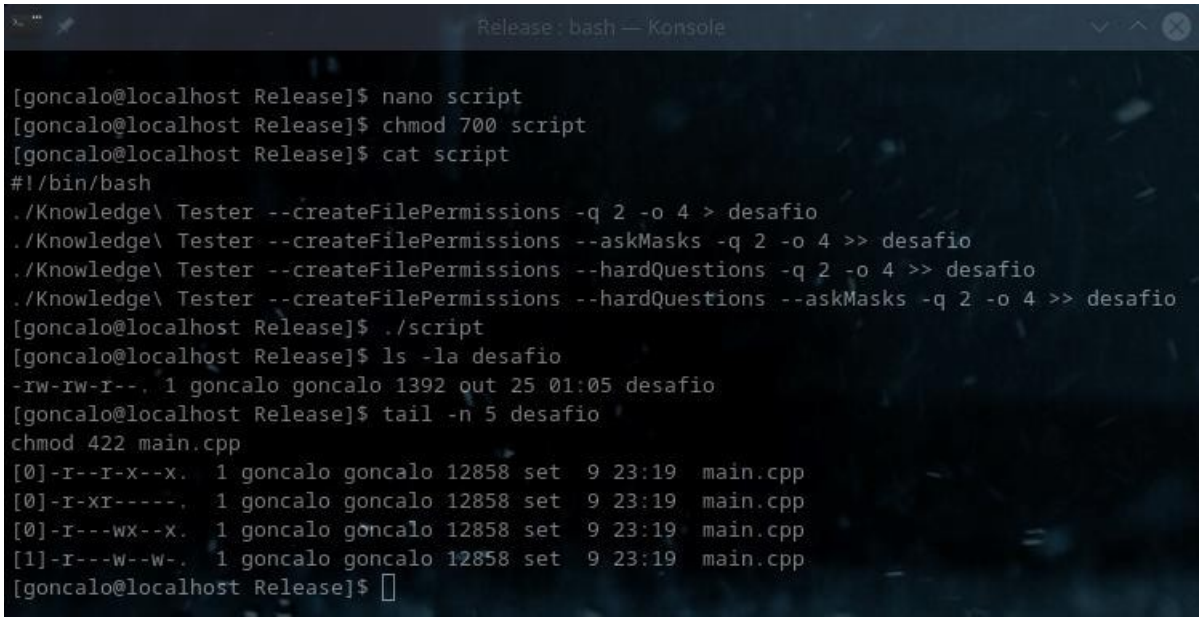
Teremos de usar quatro comandos distintos, para adicionar as perguntas ao mesmo ficheiro, e podemos até criar um script que faça isto por nós, mas os comandos são:

- ./Knowledge\ Tester --createFilePermissions -q 2 -o 4 > desafio
- ./Knowledge\ Tester --createFilePermissions --askMasks -q 2 -o 4 >> desafio
- ./Knowledge\ Tester --createFilePermissions --hardQuestions -q 2 -o 4 >> desafio
- ./Knowledge\ Tester --createFilePermissions --hardQuestions --askMasks -q 2 -o 4 >> desafio

Com a primeira linha, criamos o ficheiro “desafio”, e se já existir, é limpo primeiro e guardadas lá as perguntas depois, porque temos um único sinal de “>” após a linha de comandos.

Com as restantes linhas, já temos não um mas dois sinais de maior (“>>”), o que faz com que as perguntas sejam adicionadas ao fim do ficheiro “desafio” já existente ao invés de o sobrescrever.

Com estas quatro linhas criamos um *script* que trata da tarefa sempre que quisermos:



Criamos o script com o que podemos ver acima (visualizado o seu conteúdo com o comando "*cat script*" acima, e após lhe darmos privilégios de execução com um "*chmod 700 script*", é executado com um "*./script*", e o ficheiro "desafio" é criado com as perguntas e respostas que podem ver acima.

E assim podem juntar vários exercícios destes num só ficheiro.

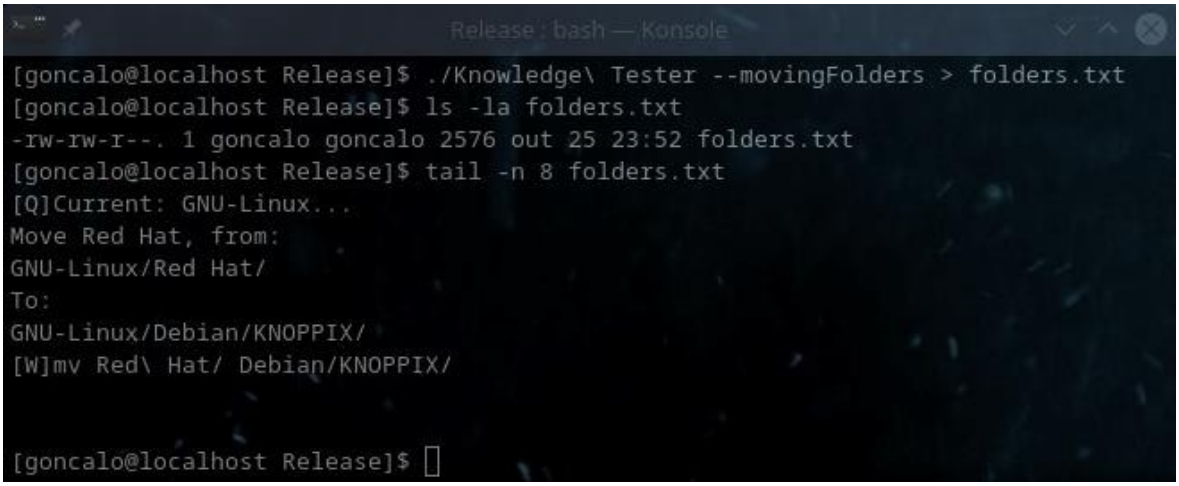
Exercícios de Mover Pastas

Este *software* tem incluído uma funcionalidade de geração de exercícios de mover pastas, para o utilizador se adaptar a guiar-se através de caminhos relativos ao copiar e mover pastas e ficheiros.

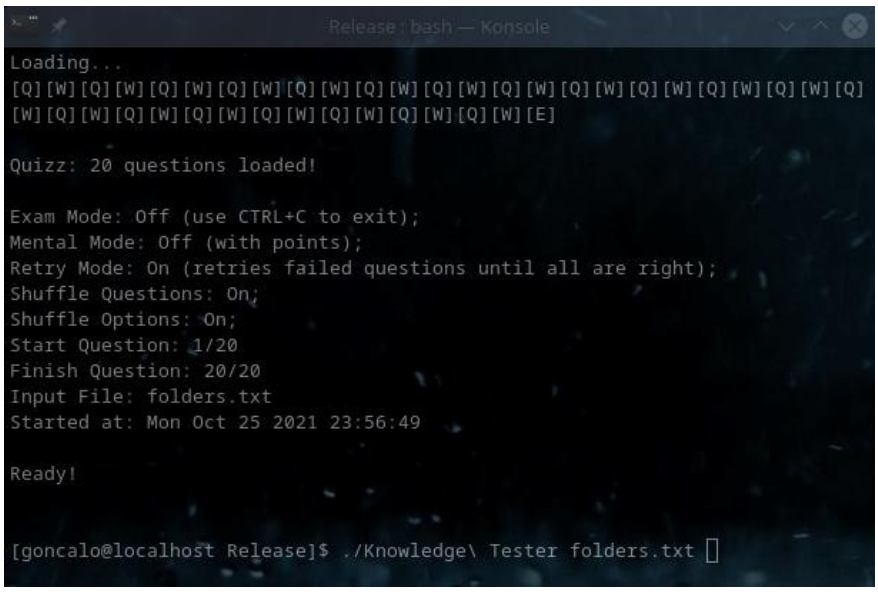
O objectivo é que o utilizador ganhe em poucos minutos a prática que levaria largas horas a ganhar, ou até dias, numa máquina real (pois numa máquina real não estaria a mover dezenas de pastas como está neste exercício), e assim acelera a aprendizagem como acontece com o exercício das *file permissions*.

Como criar exercício de mover pastas

Esse exercício é gerado através da *tag* "*--movingFolders*", e terá obviamente de ter adicionado ao fim da linha de comandos um "*> fich.txt*", onde "*fich.txt*" é o nome do ficheiro que vai conter o exame:



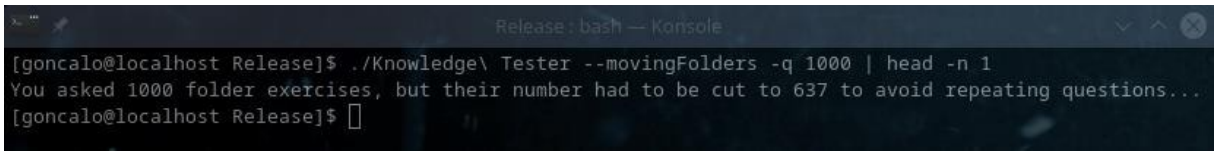
A partir desse momento, corremos o exercício como qualquer outro, com um simples "*./Knowledge\ Tester fich.txt*", ou neste caso "*folders.txt*":



Escusado será dizer que cada vez que a *tag* "*--movingFolders*" é chamada, os exercícios gerados serão completamente diferentes dos anteriores, e "aleatórios" (dentro do possível).

Opções disponíveis

Neste exercício existe a opção de alterar o número de perguntas a serem geradas, através da *tag* "*-n*", mas que está limitado ao número de 637 porque com um número superior a esse, perguntas repetir-se-iam:

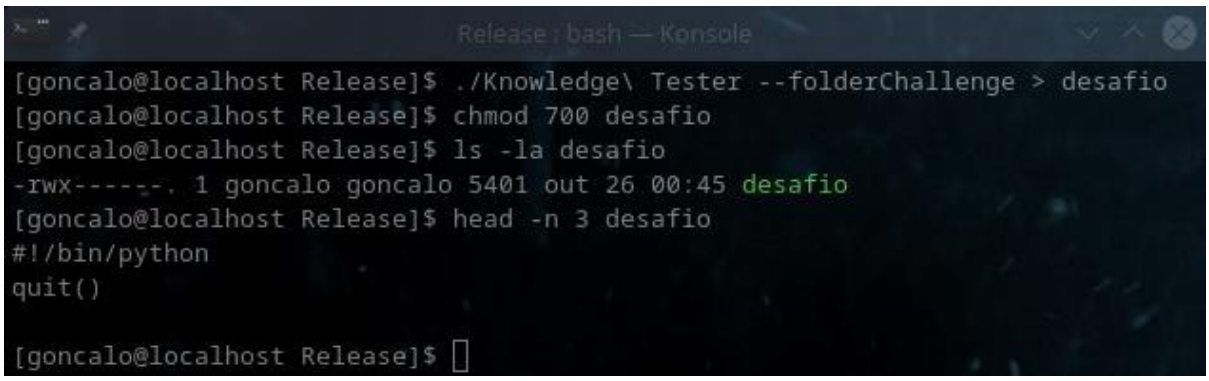


Esta é a única opção de momento disponível neste exercício…

Desafio de Pastas "*Folder Challenge*"

Este é um exercício mais complexo e trabalhoso, e que envolve não só prática a mudar nomes a pastas e movê-las, mas também algum ligeiro "*troubleshooting*", pois obriga o utilizador a resolver um pequeno problema, além de andar a verificar que pastas estão no local errado ou com nome errado, sendo um trabalho talvez mais aborrecido mas eficaz.

Este exercício invoca-se com um "*--folderChallenge*", e temos, tal como nos outros, de reencaminhar o *output* do *software* para um ficheiro de texto com o uso do sinal ">" no fim da linha, neste caso um *script* de nome "desafio":



O processo inicial é o descrito acima, o envio do *output* do comando para o *script* de nome "desafio", dar permissões de execução ao mesmo (com o "*chmod 700*" acima), e podemos ver com o comando "*head -n 3*" do exemplo acima, que o *script* é criado.

O erro de scripts a ser corrigido

Mas à partida tem logo dois erros que devem ser corrigidos:

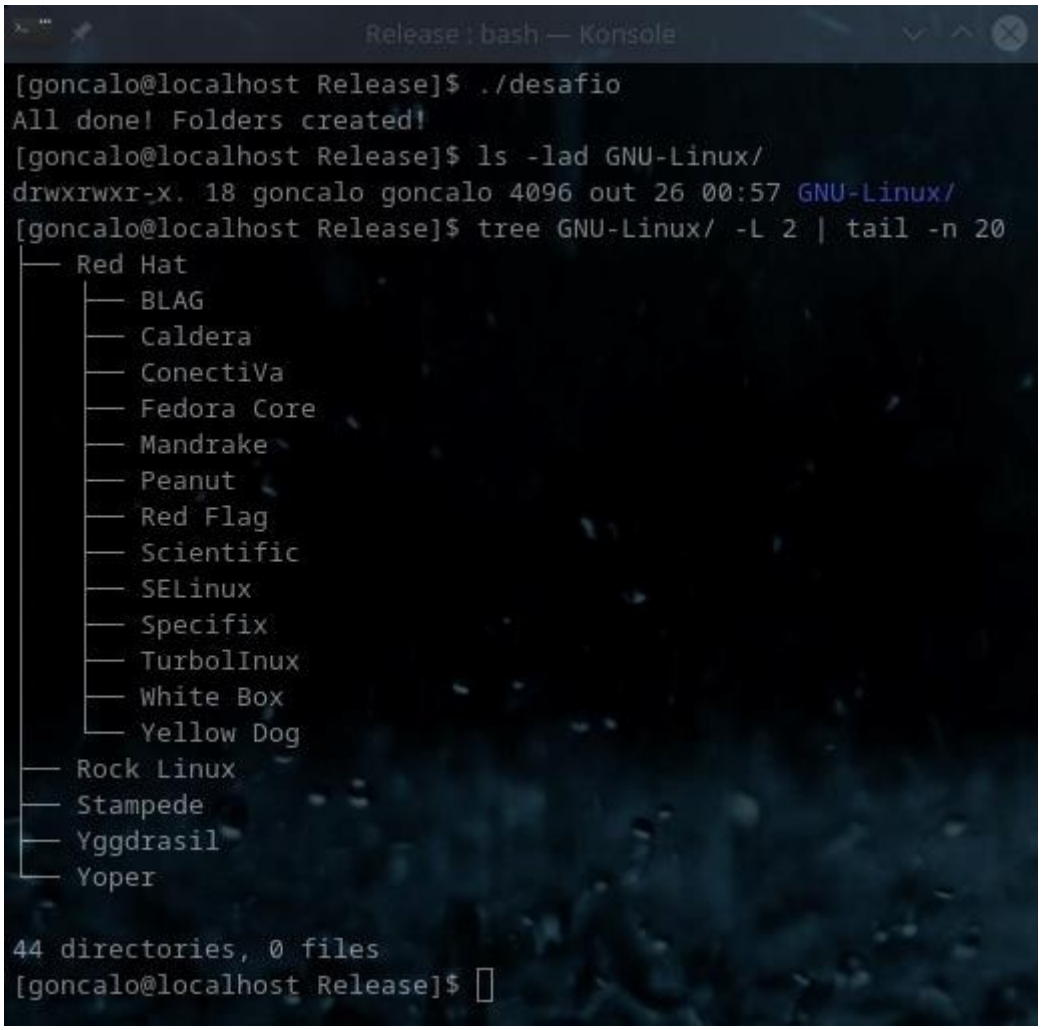
- A primeira linha deveria ter "*#!/bin/bash*" e não "*#!/bin/python*", pois o *script* é de *bash* e não de *python*. Esta linha foi incluída num exercício inicial, e acabou por tradicionalmente ficar cá, para fazer os formandos compreender para que servem.

- A linha que contém o "*quit()*" deve ser apagada, pois é um comando de *python* indesejado ao *script bash* que vamos executar.

Update: Em 2023 ao rever isto, vi que nunca mais me lembrei disto, fica para 2025! :)

A árvore de pastas criadas

Depois de corrigidos os erros, o *script* é executado com um simples "*./desafio*", e cria-nos de imediato uma pasta cheia de perto de 90 directorias com nomes de distribuições de *Linux* famosas e até as suas hierarquias, que serão as usadas para este exercício:



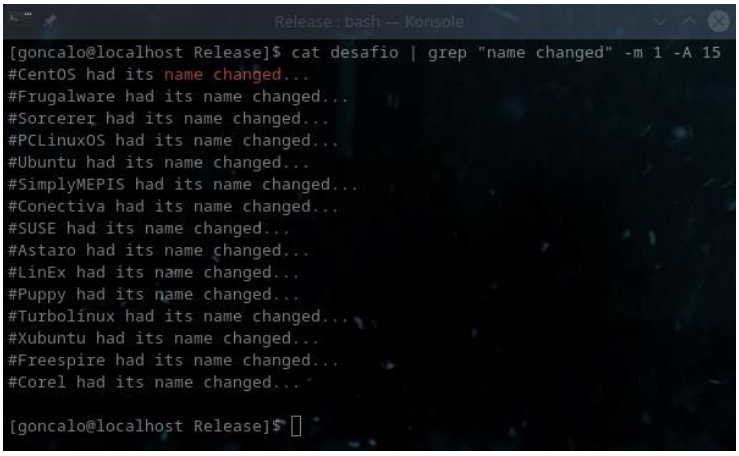
Acima podemos ver apenas até dois níveis de profundidade (com um "*head -L 2*") e as últimas 20 linhas (com um "*tail -n 20*") da árvore das perto de 90 pastas de *distros Linux*.

Podemos ver também pelas pastas acima que o *script* foi criado de forma a não só criar as pastas, como mudar o nome a várias delas (como podem ver pela imagem acima onde várias têm os seus nomes alterados) e até mover pastas de sítio.

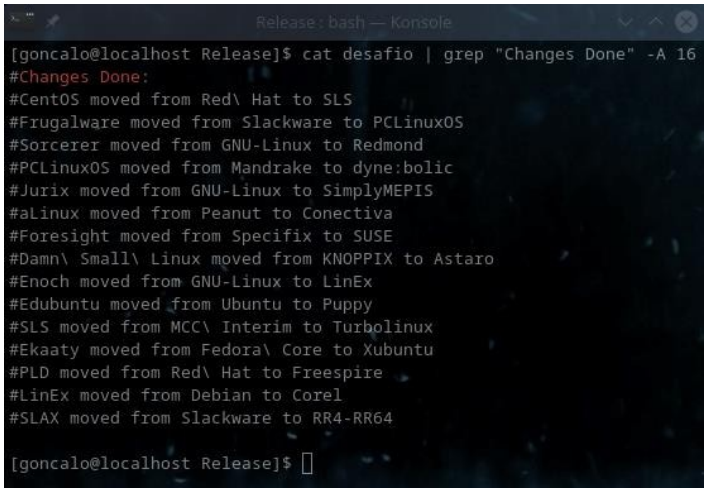
O objectivo de quem resolve o exercício é o de mudar os nomes das pastas para os nomes originais, e mover as pastas deslocadas para as suas posições originais, e isto obriga a algum treino forçado nestas tarefas.

As ajudas dadas aos formandos

O próprio *script*, como forma de ajuda aos alunos, tem a lista de pastas que mudaram de nome, para evitar que percam tempo a tentar descobrir quais foram:



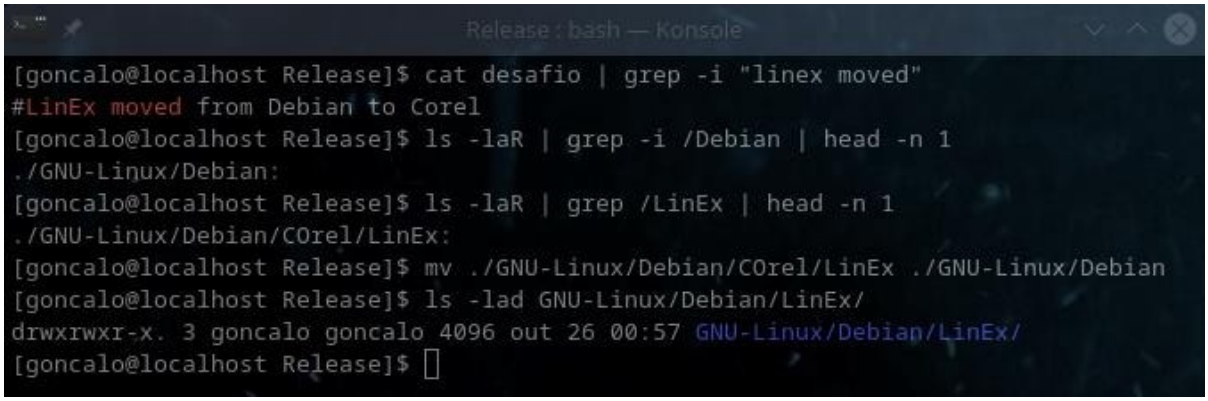
E da mesma forma, estão no ficheiro “desafio” também incluídas as mudanças de localização das pastas que foram deslocadas, para que os utilizadores saibam quais devem corrigir, onde só terão de mover da posição actual para a antiga localização:



O utilizador pode poupar tempo, com uma pesquisa rápida, com um "*grep -i*", cujo "*-i*" torna a busca "*case insensitive*" e com isso, resistente às alterações de minúsculas para maiúsculas (e vice-versa), e com uma *regular expression* de "*^GNU.*/Linex:$*", que significará algo como "linhas começadas em *GNU*, com qualquer coisa no meio, e terminadas em *Linex*:" (mas *case insensitive*), e surgirá a localização da mesma, e com um simples *move* (comando "*mv*"), é de imediato corrigida a situação:

O mover pastas de sítio, deve ser efectuado depois, e a começar de baixo para cima, para evitar problemas, pois se for noutra ordem o exercício provavelmente irá falhar.

Nessa correcção de localização de pastas, o raciocínio é simples: além de começar do último para o primeiro, só temos de localizar a actual posição da pasta a mover, que é a "*LinEx*" neste caso, e a localização da antiga pasta de origem, que neste caso era a "*Debian*", e depois mover da actual para a antiga, é tão simples quanto isso:



E repetir os passos com todas as outras.

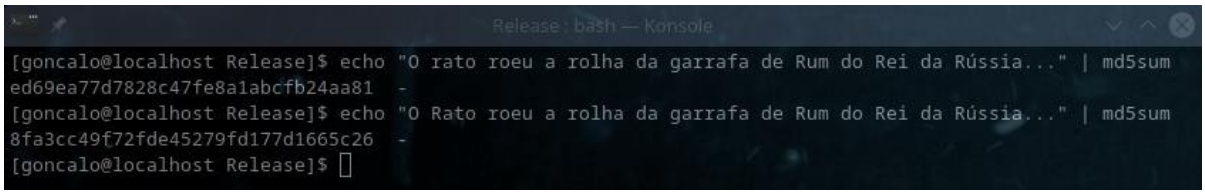
As hashes como sistema de verificação de resultados

Agora, como verificar se tudo ficou correcto?

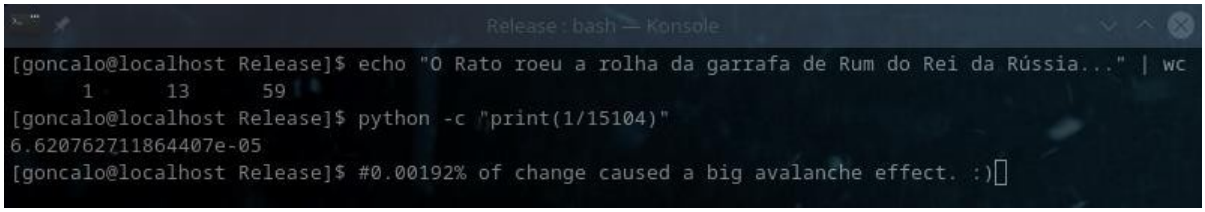
O sistema usado neste caso foi muito simples, o uso de um sistema de *hashing* criptográfico, já quebrado há muito mas mais do que suficiente para o exercício em questão, o *MD5*.

Sendo uma *hash* criptográfica, tem o chamado "efeito de avalanche", que faz com que, por muito ínfima que seja a alteração no texto que dá origem à *hash*, as repercussões na *hash* em si são sempre enormes, daí o nome "efeito avalanche".

Vejamos abaixo o "efeito de avalanche" em acção:

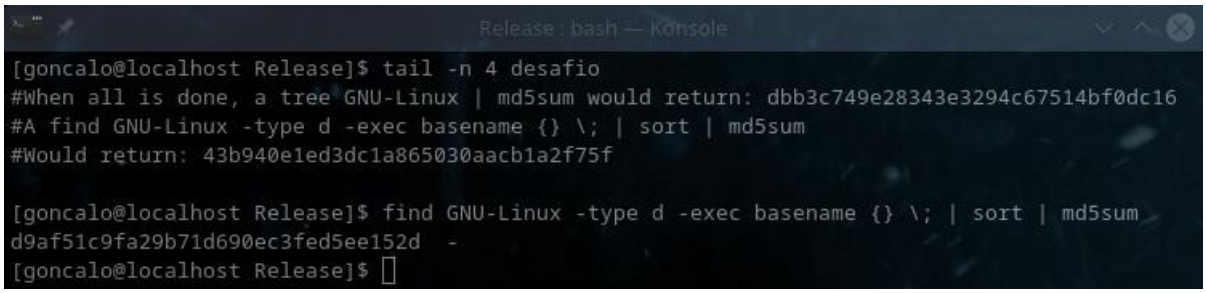


Basicamente, nestas 59 letras, uma alteração de um *bit* apenas, causaria uma alteração total da *hash MD5*, e ainda mais se for não um *bit* mas de 32, que corresponde à alteração da letra "r" para "R" no código *ASCII*:



Isto significa que uma simples letra alterada no meio dos nomes destas perto de 90 pastas que correspondem a umas 90 *distros* famosas de *Linux*, alteraria por completo a *hash*, quanto mais se estiverem pastas fora do local, etc.

Para confirmarmos, corremos o comando que é indicado no fim do ficheiro “desafio” (nome dado neste caso):



Com o comando “*find -type d -exec basename {}* \;” procuramos directorias, e temos como resultado apenas o nome das mesmas sem nada mais (graças ao comando “*basename {}* \;”), que depois são ordenados com o comando “*sort*”, e por fim gerada a *hash* correspondente.

Não sendo uma *hash* perceptual, não dá para saber a quantidade de erros que existirão pela *hash*, pois à mínima coisa, ela muda radicalmente, mas o aluno/formando terá de investigar qual o problema, até ter o resultado correcto.

E assim termina a ajuda sobre o terceito exercício embutido neste *software* de aprendizagem.

Exercícios de Programação:

Sobre os exercícios

Em 2023, criei uns quantos exercícios de Programação, em várias linguagens distintas, que funcionam de forma inversa, ou seja, o examinado tem de compreender o código em si, mesmo de cabeça, e saber resolvê-lo, e isto obriga a que a pessoa só consiga ter a resposta certa e seguir em frente, se tiver de facto uma compreensão minimamente boa sobre o que está a fazer.

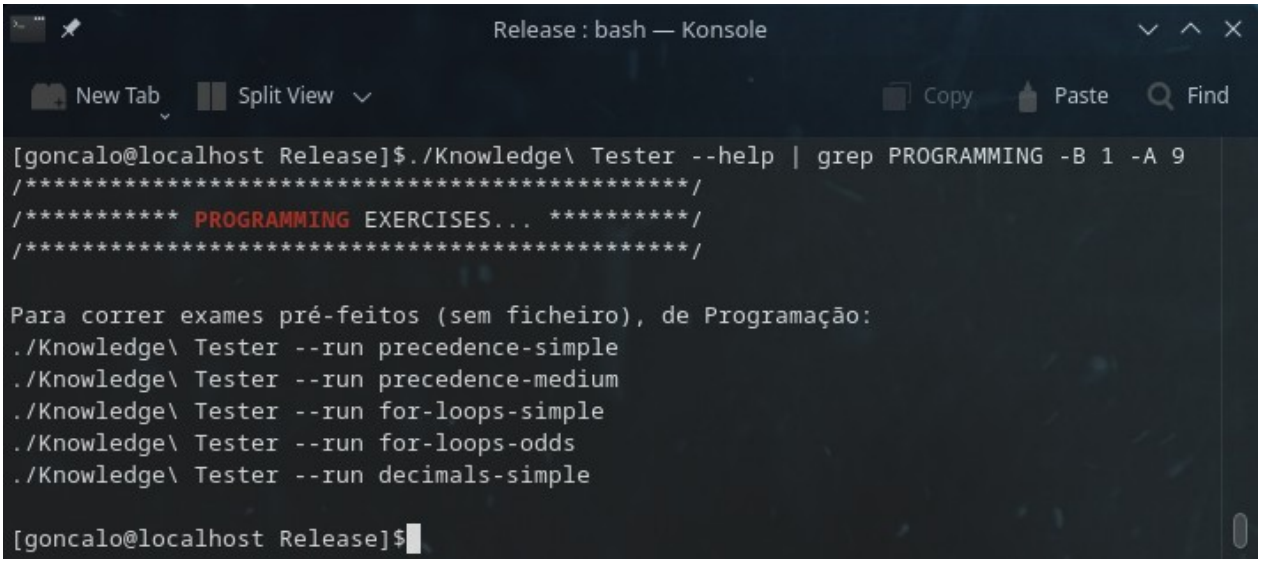
Como não dá para copiar e ir à *Internet* ver que resultados daria, pois teriam de escrever o código do zero e perderiam tempo e nota no exame, eles são mesmo forçados a saber a matéria.

E é bom para eles, pois quem souber resolver estas coisas de cabeça, já saberá programar um bocado melhor do que antes de os fazer.

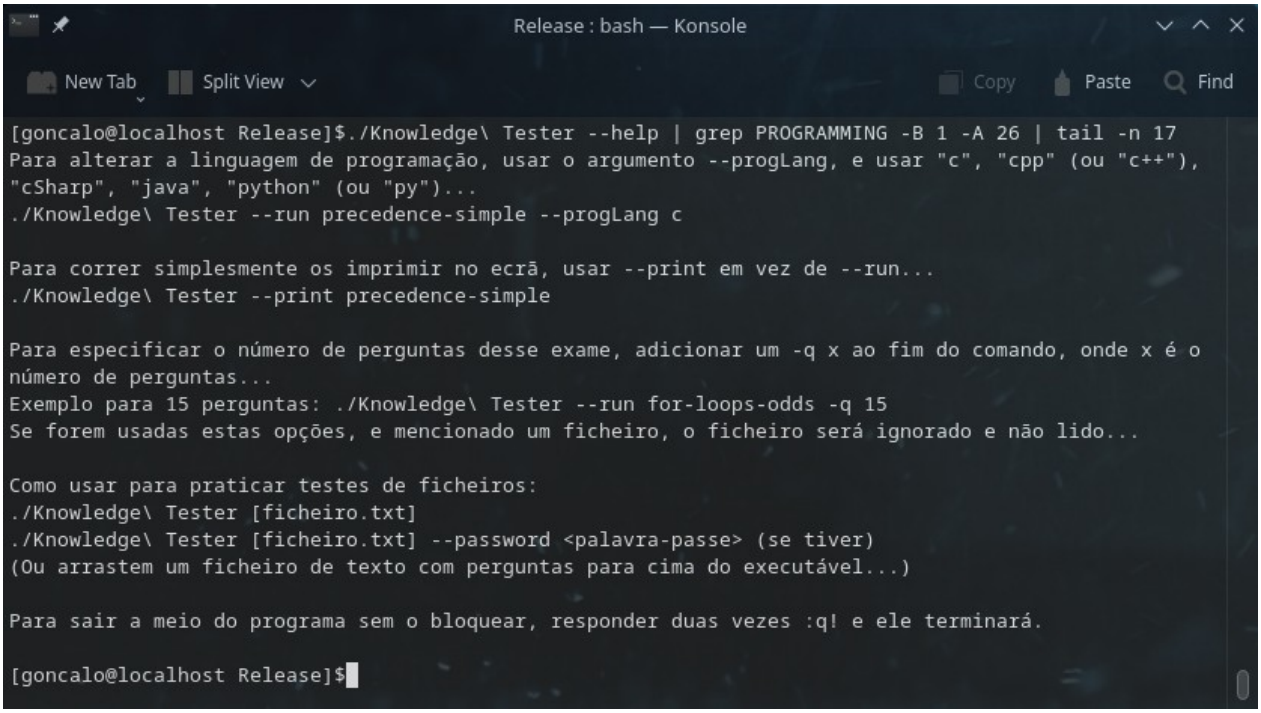
O meu *software*, sabe criar algoritmos e resolvê-los, alguns que inseri, claro está, e já tem 5 distintos.

O objectivo é um dia ir acrescentando mais, de forma a que tenha centenas de exercícios diferentes, para cada linguagem, e em que possamos simplesmente escolher a linguagem e o tipo de exercícios (podendo ter vários dentro), ou uma lista de 100 exercícios dos mais variados, e criar assim exames com minhentos tipos de exercícios diferentes, e obrigando a quem resolve estes exames a realmente compreender o código para seguir em frente.

Vejamos uma lista de exercícios disponíveis, que podem consultar através da ajuda do programa, como podem ver abaixo:



Há mais ajuda sobre este *software* no que diz respeito à ajuda:



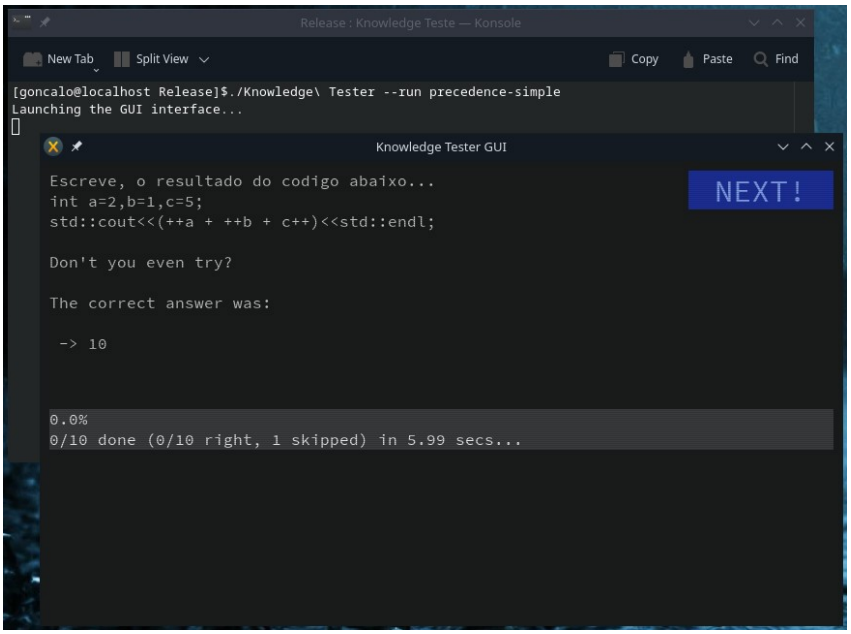
Ignorem os *greps* acima, qualquer pessoa que saiba um mínimo sobre *Linux* saberá o que fazem, se quiserem saber, investiguem na *Internet* que aprender faz bem.

Mas podemos ver acima que podemos alterar a linguagem de programação usada com um simples *–progLang* seguido da linguagem em si, que pode ser "*c*" (de *C*), "*c++*"/"*cpp*" (de *C++*), "*cSharp*" (de *C#*), "*java*" (de *Java*, obviamente), e "*python*"/"*py*" (de *Python*, obviamente).

Os tipos de exercícios disponíveis

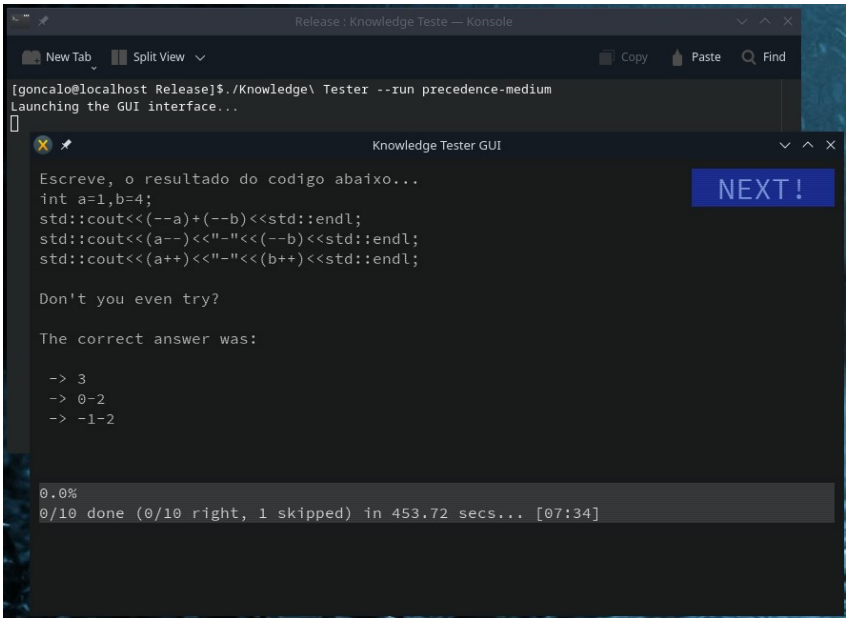
Precedência de Operadores, Simples

O primeiro é o de treino com precedência de operadores, simples:



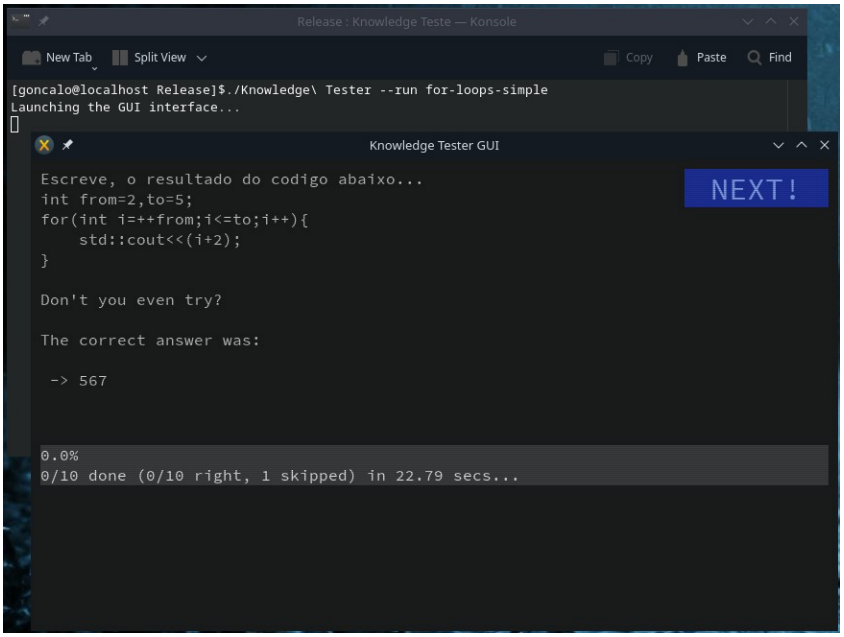
Precedência de Operadores, Médio

Depois o de treino com precedência de operadores, de média dificuldade:



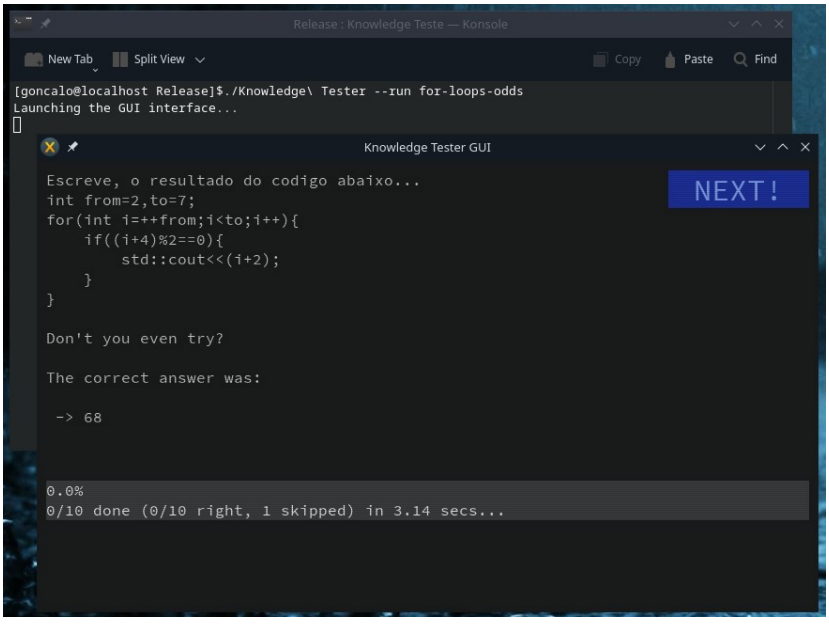
Ciclos *For*, Simples

Depois o de Ciclos *For*, simples:



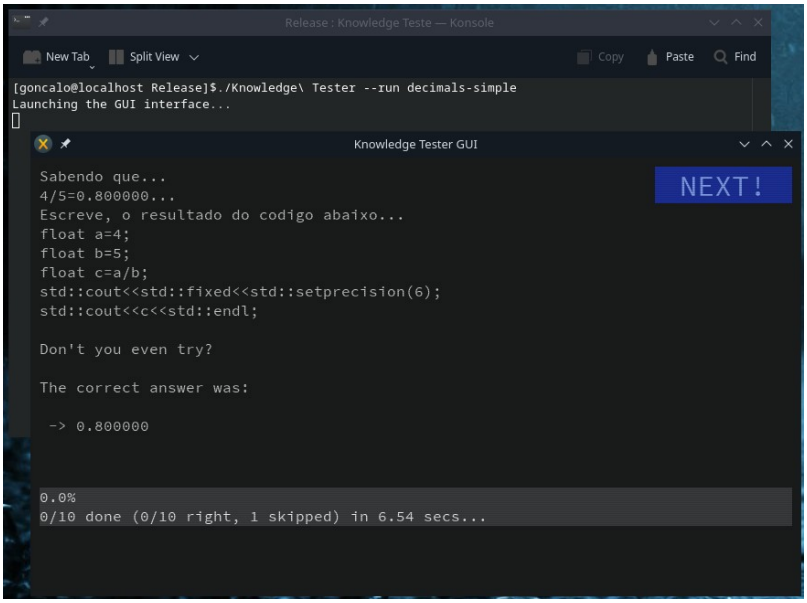
Ciclos *For*, de Dificuldade Média

Depois o de Ciclos *For*, de Dificuldade Média:



Exercícios de Casas Decimais, Simples

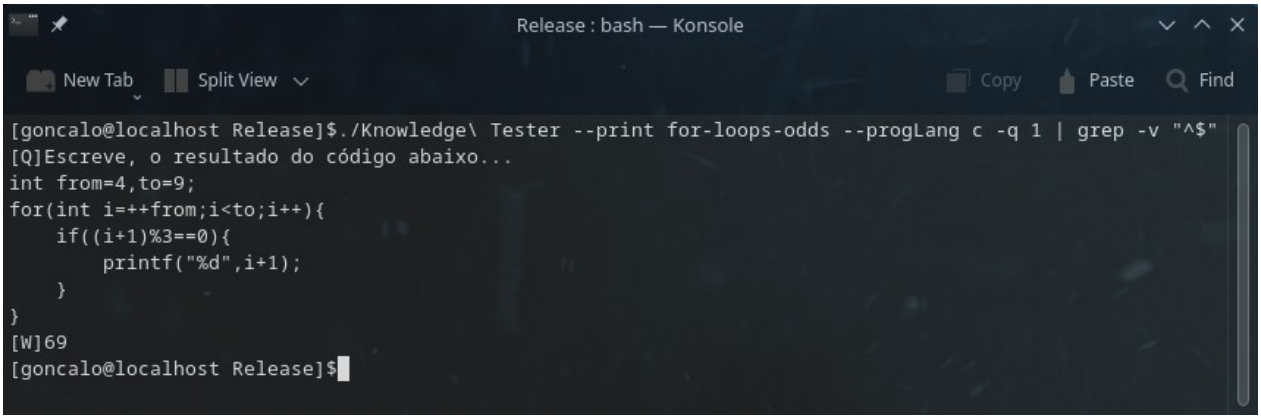
Depois os de Casas Decimais, Simples.



Escolha da Linguagem de Programação Usada:

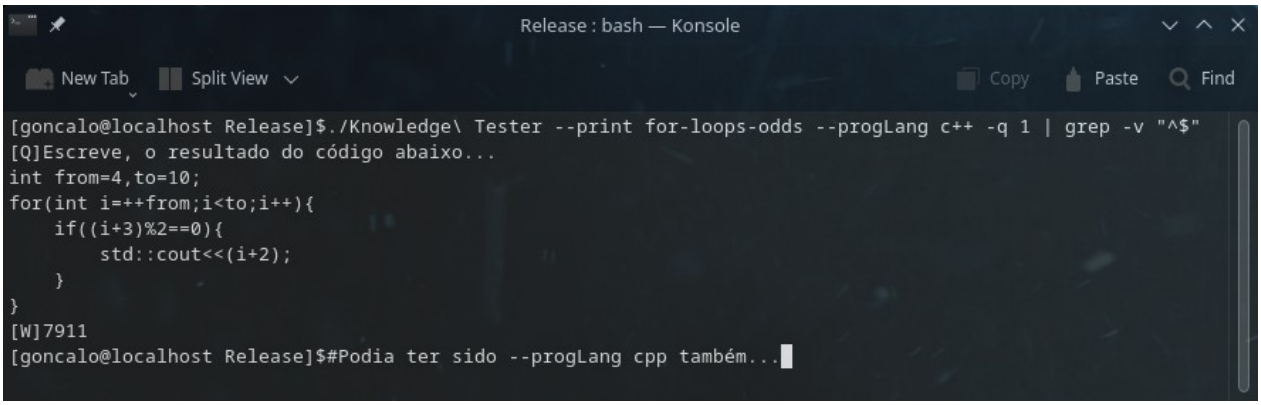
Linguagem de Programação *C*

Para a Linguagem de Programação *C*, usamos o parâmetro "*--progLang c*":



Linguagem de Programação *C++*

Para a Linguagem de Programação *C++*, usamos o parâmetro "*--progLang c++*" ou "*--progLang cpp*":



Linguagem de Programação *C#*

Para a Linguagem de Programação *C#*, usamos o parâmetro "*--progLang cSharp*":

Linguagem de Programação *JAVA*

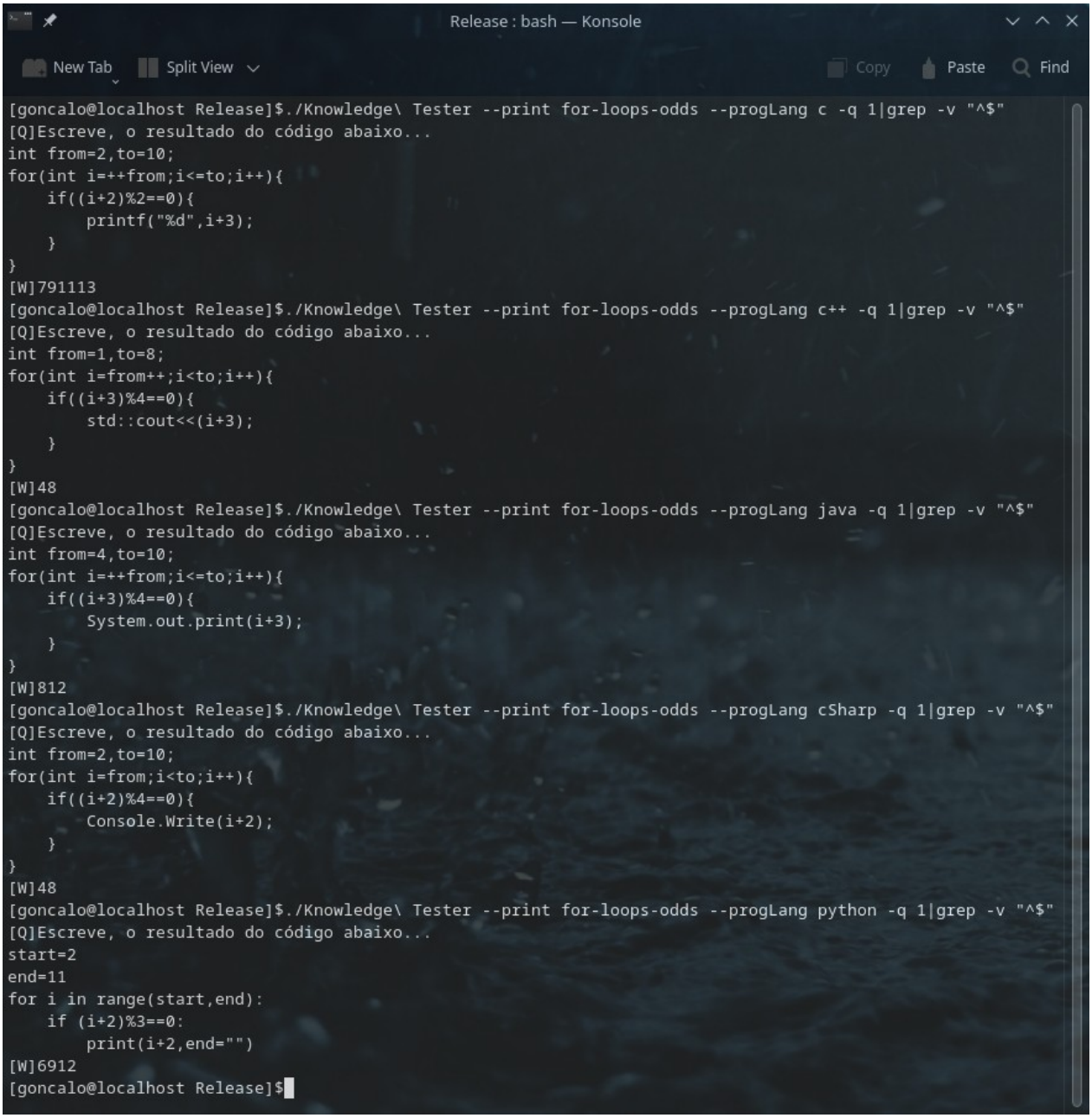
Para a Linguagem de Programação *JAVA*, usamos o parâmetro "*--progLang java*":

Linguagem de Programação *Python*

Para a Linguagem de Programação *Python*, usamos o parâmetro "*--progLang python*" ou "*--progLang py*":

Um exemplo de um exercício em todas as linguagens com o –*print*

Com o parâmetro –*print* podemos mostrar no ecrã ao invés de correr, um exercício, e fica aqui o mesmo com todas as linguagens para vermos as diferenças entre elas:



Há-de ser tudo melhorado um dia no Futuro. :)

Ideias Futuras

Deixo aqui por curiosidade o que tinha escrito em 2020/2021, com updates do que fiz em 2023, tenho muitíssimas ideias para 2025, mas são tantas que não vale a pena meter aqui tudo, mas posso dizer que a *GUI* já terá menus, e muitas opções, e será altamente, e os exames terão 1001 opções novas, encontrarão mais sobre isso em www.goncalo.pt:

2020/2021:

Provavelmente este *software* irá ficar pendente durante uns bons meses, à medida que regressarei a projectos parados há já algum tempo, como o meu *Game Engine* em *C/C++*, mas mais tarde hei-de voltar a ele.

Mas antes de ser colocado em *stand-by*, tenciono ainda em 2021:

- Criar um sistema de opções forçadas, para que possamos gerar um exame que corra obrigatoriamente com certas opções activadas e que seja impossível desligar com parâmetros na linha de comandos; **-> FEITO em 2023!!! :)**

- Criar um sistema (talvez) de respostas múltiplas por cada linha, ou seja, numa resposta multi-linha, permitir que certas linhas tenham várias respostas possíveis, talvez ainda este ano, ou talvez fique para o próximo; **-> FEITO em 2023!!! :)**

- Poder definir a pontuação de cada pergunta, ao invés de ser como agora em que cada pergunta valerá o valor de 100% a dividir pelo número de perguntas total; **-> FEITO em 2023!!! :)**

Update: Curiosamente, agora vendo em 2023, fiz isto, e MUITO mais!

De resto, para 2021, parece-me mais do que completo de momento.

No Futuro, mais tarde, pretendo:

- Criar talvez uma *GUI*, para fazer exames em modo gráfico, a imitar certificações; **-> FEITO EM 2023, ou pelo menos iniciado o projecto da *GUI*!!! :)**

- Talvez possibilitar a exibição de imagens em exames, mesmo em modo de texto (alterando directamente os *buffers* de memória usados pelo *Linux* para vídeo) em modo de terminal; **-> FEITO EM 2023!!! :)**

- Acrescentar outro tipo de perguntas e respostas, como ordenações, etc; **-> Esta ainda não fiz... :(**

- E outras coisas que não me ocorrem de momento. **-> Fiz muitas outras coisas, foi um bom ano para este *software*, 2023, agora fica para 2025!!! :)**

Contactos e Informações

Em caso de críticas construtivas, ou sugestões, ou dúvidas, contactem-me quem me conhece pelos meios tradicionais, e quem não me conhece, pelo *LinkedIn* (*http://www.linkedin.com/in/goncalopt/*), ou *email*, disponíveis no meu *website* *http://www.goncalo.pt/*.

E quando ao programa em si, bem como este manual, podem sacá-lo sempre no meu *website*, em https://www.goncalo.pt/por/projecto-knowledge-tester/.

Lá encontrarão a versão mais recente do *software*, e até do tutorial, que até ter 2025 não tocarei mais nele. :)