Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

© Pedro R. M. Inácio (inacio@di.ubi.pt), Tiago M. C. Simões (tsimoes@di.ubi.pt)
e Tiago Roxo (tiago.roxo@ubi.pt)

Laboratórios de Programação
Guia para Aula Laboratorial 5
Licenciatura em Engenharia Informática

Programming Laboratories
Guide for Laboratory Class 5
Degree in Computer Science and Engineering

Sumário
Scripting em *bash*. Compilação e execução de programas em `C`. Criação de ficheiro `Makefile`.

Summary
Bash Scripting. Compilation and execution of C programs. `Makefile` *creation.*

Pré-requisitos:
Algumas das tarefas propostas a seguir requerem o acesso à Internet e acesso a um **sistema operativo Linux**. Se não pretender instalar uma distribuição de Linux na sua máquina pode sempre optar por instalar a distribuição numa máquina virtual. O uso de Subsistema Windows para Linux (*Windows Subsystem for Linux*), para Windows 10, concretiza também uma opção válida.

1 *Scripts* em *bash*

Bash Scripts

Desenvolver *scripts* em *bash* é extremamente útil para a administração e no desenvolvimento de sistemas ou de programas. Estes ficheiros são conjuntos de comandos ordenados e com uma determinada intenção, executados sequencialmente e, potencialmente, adornados por instruções de controlo de fluxo.

Tarefa 1 *Task 1*

Crie um ficheiro, denominado `script.sh`, com o trecho de código seguinte. Modifique a variável `APELIDO` para que tenha o seu apelido.

```
#!/bin/bash
APELIDO="Inácio"
```

Q1.: Qual é a utilidade da primeira linha do ficheiro criado na tarefa anterior?
- ☐ É um comentário por isso não tem utilidade nenhuma.
- ☐ Serve para indicar que o *script* deve ser interpretado pelo *bash* e não por outro interpretador.

Tarefa 2 *Task 2*

Modifique o ficheiro de modo a que, quando executado, imprima, no terminal, a frase: `"Bom dia, Inácio!"`. Para tal deverá utilizar a variável `APELIDO`.

Q2.: Qual foi o comando que usou para imprimir a mensagem no terminal?
☐ `touch` ☐ `print` ☐ `echo` ☐ `grep` ☐ `ls`

Para utilizar o conteúdo da variável `APELIDO` no comando anterior teve que lhe adicionar um símbolo.
Q3.: Qual foi o símbolo utilizado?
☐ @ ☐ : ☐ ~ ☐ ☠ ☐ $ ☐ # ☐ > ☐ ♘

Tarefa 3 *Task 3*

Execute o ficheiro `.sh` utilizando o comando `bash script.sh`, no terminal.

Q4.: A execução do ficheiro culminou na impressão da mensagem esperada?
- ☐ Não. Apareceu `APELIDO` ao invés do meu apelido.
- ☐ Sim. O terminal deu-me os bons dias e o meu dia até ficou melhor!

Considere que queria executar o *script* usando o comando `./script.sh` **Q5.: É possível?**
- ☐ Sim. Eu executei e deu o mesmo resultado.
- ☐ Sim, é possível. Mas para tal é necessário usar o comando `chmod u+r script.sh` antes.
- ☐ Sim, é possível. Mas para tal é necessário usar o comando `chmod u+x script.sh` antes.

☐ Não. Tal coisa não é possível em terminal. Desisto.

Tarefa 4 *Task 4*

Modifique o ficheiro `script.sh` para que receba, como argumento, o seu primeiro nome e o inclua na mensagem a imprimir no terminal. Por exemplo, caso o seu primeiro nome seja `Pedro`, o *script* deverá emitir a mensagem "`Bom dia, Pedro Inácio!`"quando é executado da seguinte forma:

```
./script.sh Pedro
```

Q6.: O que teve que adicionar ao comando `echo` para que produzisse a mensagem requerida na tarefa anterior?

☐ Para usar o primeiro argumento tive que incluir `#1` no comando `echo`.
☐ Para usar o primeiro argumento tive que incluir `$1` no comando `echo`.
☐ Criei uma nova variável, chamada `NOME` e usei-a no comando `echo` de forma análoga à variável `APELIDO`.
☐ Vou confessar. Fui batoteiro e escrevi `Pedro` no comando `echo` porque já não me lembro como se usa os argumentos...

Um cuidado que deve ter, quando espera receber argumentos, é verificar que, efetivamente, estes foram dados ao *script* aquando da sua execução. No exemplo anterior, mesmo que o comando seja executado sem um argumento, o *script* é executado e é exibida a mensagem apenas com o apelido d@ estudante.

Tarefa 5 *Task 5*

Execute o seu ficheiro `script.sh`, sem argumentos, e verifique que nenhum erro é exibido no terminal.

Para recordar um conceito apresentado na aula teórica, considere o seguinte exemplo:

```
VAR=101
if [[ $VAR -gt 100 ]]
then
  echo "O número é superior a 100."
else
  echo "O número é inferior a 100."
fi
```

Q7.: Qual é a finalidade de `-gt`, no trecho de código exibido?

☐ É um argumento de *if* que aumenta o valor (*give them*) da variável `VAR` em 100 unidades.
☐ É uma opção passada à variável `$VAR` para obter o valor desta (*get that*) e, neste exemplo, substituir por 100.
☐ É um operador de comparação. Neste caso, é usado para que a condição *if* sirva para verificar se o valor de `VAR` é superior (*greater than*) a 100.
☐ Nem vi que estava um `-gt` no trecho de código. Deixe-me só ver melhor e já lhe dou uma resposta.

Tarefa 6 *Task 6*

Seguindo a analogia do uso de `-gt` no exemplo anterior, modifique o ficheiro `script.sh` para que seja verificado o número de argumentos e, caso o número recebido **não seja concordante com o esperado**, o ficheiro deverá imprimir, no terminal, a mensagem "`Número incorreto de argumentos!`". Para verificar o número de argumentos recebidos, use a seguinte combinação de caracteres: `$#`.

Execute novamente o ficheiro *script.sh* sem argumentos. **Q8.: Houve alguma alteração relativamente à execução, deste mesmo ficheiro, comparativamente à tarefa 5?**

☐ Nenhuma. Em ambos os casos apresenta uma mensagem de bom dia no terminal.
☐ Houve, sim senhor! Agora o *script.sh* exige que lhe passe um argumento. Será que se der mais do que um também funciona?

A leitura de *inputs* do utilizador e armazenamento destes numa variável é outra funcionalidade que a *bash* disponibiliza. **Q9.: Qual é o comando que pode utilizar para usufruir desta funcionalidade?**

☐ `get` ☐ `scanf` ☐ `nano` ☐ `read` ☐ `stdin`

Tarefa 7 *Task 7*

Altere o ficheiro `script.sh` para que, dentro da condição *if* e depois da mensagem de bom dia, imprima a mensagem "`Introduza a sua idade:`", leia o *input* do utilizador para uma variável, denominada `IDADE`, e imprima uma segunda mensagem, indicando a idade do utilizador. Caso o utilizador introduza o valor `18`, a mensagem a imprimir deverá ser "`Tem 18 anos`".

Só para verificar que todos os conceitos usados ficaram consolidados: **Q10.: qual o comando que usou para ler o *input* do utilizador e guardá-lo na variável `IDADE`?**

☐ `read $IDADE` ☐ `read IDADE`
☐ `read #IDADE` ☐ `read -IDADE`

E para que não restem mesmo dúvidas nenhumas: **Q11.: qual o comando que usou para imprimir a mensagem com a variável** `IDADE`**?**

☐ `echo "Tem $IDADE anos!"`
☐ `echo "Tem IDADE anos!"`
☐ `echo "Tem #IDADE anos!"`
☐ `echo "Tem -IDADE anos!"`

2 Edição, Compilação e Execução de Programas

Program Editing, Compilation and Execution

Para o computador executar programas é necessário que os mesmos estejam escritos em linguagem máquina, ou que exista um interpretador que faça a tradução em tempo-real. Assim, é necessário que exista uma tradução entre o que o programador escreveu e o que a máquina compreende. O desenvolvimento de um programa, fazendo uso de linguagens compiladas, passa por três fases essenciais:

1. Implementação do código;
2. Compilação e interligação do código;
3. Execução e teste do código.

Considere o comando seguinte: `cc -o main.exe main.c`. **Q12.: Em qual das fases anteriormente apresentadas se enquadraria o comando apresentado?**

☐ Implementação do código.
☐ Compilação e interligação do código.
☐ Execução e teste do código.
☐ Atenção malta! Pergunta com rasteira! Não pertence a nenhuma das fases.

Q13.: O que pode comentar relativamente ao uso da opção `-o` no comando apresentado?

☐ É uma opção facultativa que permite definir o nome do executável.
☐ É uma opção obrigatória que permite a compilação dos ficheiros `.c` em ficheiro `.o` .

Tarefa 8 *Task 8*

Obtenha um ficheiro `.zip` usando esta *hiperligação*. Faça `unzip` e confirme a existência de dois ficheiros `.c` e um ficheiro `.h`. Coloque os três ficheiros numa nova diretoria, denominada `Lab_5_Compile_Make`. Analise os ficheiros e interprete a funcionalidade de cada um deles.

Só para recordar: **Q14.: qual é o significado de ficheiros com extensão** `.h`**?**

☐ Significa que é ficheiro de ajuda (*helper*).
☐ Significa que é ficheiro de cabeçalho (*header*).
☐ Significa que é ficheiro de ligação (*holder*).
☐ Significa que é ficheiro de excitação (*hype*)!

Observe, atentamente, o ficheiro `main.c`. **Q15.: Qual é a utilidade da inclusão do ficheiro** `soma.h`**?**

☐ Permite que seja usada uma função que está presente no ficheiro `soma.c`.
☐ É uma gralha. Devia ter sido incluído o ficheiro `soma.c` e não `soma.h`.

Ainda relativamente ao ficheiro `main.c`. **Q16.: Qual é a razão para que** `soma.h` **tenha sido incluído entre " ", enquanto** `stdio.h` **foi incluído com** < >**?**

☐ Foi uma opção do programador. Ambos podem ser usados de forma intercambiável.
☐ Ambos são ficheiros de cabeçalho mas `soma.h` é ficheiro de cabeçalho local enquanto `stdio.h` é do compilador.

Tarefa 9 *Task 9*

Registe qual é a finalidade do programa, descriminando o papel desempenhado por cada um dos ficheiros.

Considere agora o processo de compilação do código. **Q17.: Qual o comando que utilizaria para compilar o programa descrito na tarefa anterior?**

☐ `cc` ☐ `gcc` ☐ `g+-` ☐ `java` ☐ `compilador_c`

Q18.: Caso a sua intenção fosse compilar os ficheiros e obter os ficheiros objetos destes (com extensão `.o`**) qual seria a opção a usar, aquando do processo de compilação?**

☐ `-o` ☐ `-c` ☐ `-v` ☐ `-x` ☐ `-t`

Assuma que pretende compilar os ficheiros fonte fornecidos e obter um executável. **Q19.: Qual, das seguintes opções, permitiria atingir esse objetivo?**

☐ `cc main.c` ☐ `cc main.c soma.h`
☐ `cc soma.c soma.h` ☐ `cc main.c soma.c`

Q20.: Dada a opção escolhida na questão ante-

rior, de que forma executaria o ficheiro executável resultante?

- ☐ `./main.c`
- ☐ `./main.exe`
- ☐ `./a.out`
- ☐ `./a.exe`
- ☐ `./nao_faco_ideia.exe`

Tarefa 10 ***Task 10***

Registe o comando que utilizaria para que a compilação dos ficheiros fonte fornecidos resultasse num ficheiro executável com o nome `eu_e_que_mando.exe`.

Execute o ficheiro resultante e confirme que produz o resultado esperado.

3 Makefile

Makefile

O `Make` é uma ferramenta de engenharia de *software* que controla a geração de executáveis e outros ficheiros associados a partir dos ficheiros fonte de um programa. A sua utilização ganha maior pertinência com o aumento do número de componentes que contribuem para um programa. Adicionalmente, permite diminuir a probabilidade de erro humano no processo de compilação.

Relembre a sintaxe das regras de um ficheiro `Makefile`, abordada na aula teórica, e considere o trecho de código seguinte:

```
main.o : main.c
    cc -c main.c
```

Q21.: Qual é a designação de `main.o` e `main.c` na nomenclatura dos `Makefiles`, respetivamente, no exemplo anterior?

- ☐ Objetivo e Comando.
- ☐ Objetivo e Dependência.
- ☐ Dependência e Objetivo.
- ☐ Comando e Objetivo.

E para confirmar que percebeu os conceitos: **Q22.: qual é a designação de `cc -c main.c` no trecho de código apresentado?**

- ☐ Objetivo.
- ☐ Dependência.
- ☐ Comando.
- ☐ Não sei e recuso-me a responder, e por isso também não escolho esta opção...

Q23.: Qual é a finalidade da regra apresentada no trecho de código?

- ☐ É uma regra que necessita de um ficheiro `main.o` e que, caso este exista e esteja acedível, executa um comando que permite obter o ficheiro fonte de `main`, ou seja, `main.c`.
- ☐ É uma regra que necessita de um ficheiro `main.c` e que, caso este exista e esteja acedível, executa um comando que permite obter o ficheiro objeto de `main`, ou seja, `main.o`.
- ☐ É uma regra que necessita do comando `cc -c main.c` e que cria o ficheiro `main.o`, independentemente de `main.c` existir ou não.
- ☐ É uma regra confusa e desnecessária que só está aqui para me confundir e eu já não estou a perceber nada disto...

Tarefa 11 ***Task 11***

Obtenha um ficheiro `Makefile` usando esta *hiperligação*. Coloque este ficheiro na diretoria `Lab_5_Compile_Make`, criada numa tarefa anterior. Analise o ficheiro e interprete a funcionalidade de cada uma das regras nele contidas.

Q24.: Assumindo que tem os ficheiros `.c`, `.h` e `Makefile` disponibilizados na diretoria `Lab_5_Compile_Make`, o que prevê que aconteça caso execute o comando `make`?

- ☐ Será executado apenas a primeira regra, `all`, que necessita de um ficheiro `main.exe`. No entanto, como este ficheiro não existe, será emitido uma mensagem de erro no terminal.
- ☐ Tenho uma vaga ideia mas estou só à espera que o Professor faça para depois poder responder com mais certeza.
- ☐ Serão executados um conjunto de comandos e no final será criado um executável, denominado `main.exe`.
- ☐ Serão executados um conjunto de comandos mas, seguindo as regras todas, no final teremos a regra `clean` o que significa que toda a diretoria será limpa e ficará sem ficheiros.

Uma boa prática na construção de um ficheiro `Makefile` passa por identificar quais são os *phony targets* e colocá-los numa entrada, denominada `.PHONY`.

Q25.: O que pode comentar relativamente a um *phony target*?

- ☐ Não produz nenhum ficheiro.
- ☐ Não tem nenhuma dependência.
- ☐ Não tem nenhuma funcionalidade.

☐ Não faz bem o que é suposto. É falso!

Tarefa 12 *Task 12*

Coloque todos os *phony targets* do ficheiro `Makefile` numa entrada `.PHONY`.

Q26.: Quantos *phony targets* colocou na entrada `.PHONY` ?
☐ zero ☐ um ☐ dois ☐ três ☐ quatro ☐ $7 \pm \sqrt{2}$

Tarefa 13 *Task 13*

Modifique o ficheiro `Makefile` para que todos os comandos `cc`, de todas as regras, sejam substituídos por uma variável `CC`, cujo conteúdo é `cc`.

Q27.: Qual foi a aparência do comando `cc -c main.c` no `Makefile`, após realizar as alterações pedidas?
☐ `#(CC) -c main.c` ☐ `#(cc) -c main.c`
☐ `$(cc) -c main.c` ☐ `$(CC) -c main.c`

Note que a execução do comando `make` produz o ficheiro `main.exe` e que, para o executar, terá que executar o comando `./main.exe`.

Tarefa 14 *Task 14*

Assumindo que percebeu o funcionamento do ficheiro `Makefile`, modifique-o de modo a que, **apenas executando o comando `make`** este compile, faça *linkagem* e execute o ficheiro resultante. Como sugestão, crie uma nova regra, com o objetivo `execute` e atribua as dependências e comandos necessários.

Q28.: Qual foi a dependência que associou à regra criada ou, por outras palavras, do que é que precisa para executar o programa?
☐ `main.exe` ☐ `./main.exe`
☐ `execute` ☐ `main.c`

Q29.: Alterou algo no objetivo `all`?
☐ Não... Era suposto?
☐ Sim, tive que substituir `main.exe` por `execute`.

Q30.: E a entrada `.PHONY`, foi alterada?
☐ Tive dúvidas, mas coloquei lá o `execute`.
☐ Não, pois `execute` não é um *phony target*.

Recorde a utilização de variáveis automáticas, lecionado na aula teórica. Tome particular atenção à *wildcard* `%`, cujo o significado é: *qualquer sequência de caracteres*. Observe também a variável `<`, que, em conjunto com `$`, ou seja, `$<` significa: *nome da primeira dependência*.

Tarefa 15 *Task 15*

Altere o ficheiro `Makefile` para que todas as regras que tenham um objetivo com extensão `.o` sejam substituídas por uma única regra que faça uso de `%`, tanto no objetivo como na dependência, e que use `$<` no comando a executar.

Q31.: Como ficou o objetivo e a dependência da nova regra criada?
☐ `%.c: %.o` ☐ `%.o: %.c` ☐ `%.exe: %.o`

Q32.: Quantas regras foram eliminadas, com a inserção desta nova?
☐ $-7 \pm \sqrt{2}$ ☐ zero ☐ uma ☐ duas ☐ três ☐ quatro

Tarefa 16 *Task 16*

Introduza o trecho de código seguinte no topo do seu ficheiro `Makefile`

```
SOURCEFILES = $(wildcard *.c) # Forma de obter
    os nomes de ficheiros .c. Neste caso,
    todos os ficheiros com extensão .c serão
    armazenados na variável SOURCEFILES
OBJECTS = $(SOURCEFILES:%.c=%.o) # Substitui,
    em cada ficheiro da lista de ficheiros em
    SOURCEFILES, a extensão .c por .o
```

e analise os comentários.

Q33.: O que ficou armazenado na variável `OBJECTS`?
☐ Todos os ficheiros da diretoria que já tinham extensão `.o` .
☐ Um conjunto de ficheiros com extensão `.o`, cujos nomes correspondem a ficheiros da diretoria que têm extensão `.c` .

Recorde, novamente, a utilização de variáveis automáticas em `Makefile`. A utilização da variável `^`, que, em conjunto com `$`, ou seja, `$^` significa: *conteúdo da lista de dependências*.

Atente, no ficheiro `Makefile`, à regra

```
main.exe : main.o soma.o
  $(CC) -o main.exe main.o soma.o
```

e verifique que, tanto nas dependências como no comando, `main.o` e `soma.o` estão presentes. Na realidade, o conteúdo da variável `OBJECTS` é precisamente estes dois ficheiros.

Tarefa 17 ***Task 17***

Modifique a regra apresentada, no ficheiro `Makefile`, para que faça uso da variável `OBJECTS` e de `$^`.

Só para verificar que fez as alterações corretas: **Q34.: o que colocou nas dependências da regra alterada?**

☐ `$^` ☐ `$(OBJECTS)` ☐ `$(OBJECTS) $^` ☐ `./$^`

Relativamente à utilização de variáveis automáticas em `Makefile`, a utilização da variável `@`, que, em conjunto com `$`, ou seja, `$@` significa: *nome do objetivo da regra*. Com um olhar mais atento, verifica-se que `main.exe` está no comando da regra apresentada e é, simultaneamente, o objetivo da regra.

Q35.: Acha que conseguiria fazer uso de `$@` no comando da regra apresentada?

☐ Conseguir, conseguia. Mas é melhor não mudar mais não vá isto deixar de funcionar...

☐ Sim, substituiria `main.exe` por `$@` no comando da regra.

Tarefa 18 ***Task 18***

Se fez todas as tarefas corretamente até agora, irá verificar que `main.exe` aparece três vezes no seu ficheiro `Makefile`.

Tendo em conta todas as opções que lhe foram apresentadas ao longo deste guia, faça todas as alterações que achar conveniente para que `main.exe` apareça apenas uma vez no ficheiro `Makefile`.