Tarefa 0 - Primeiros passos

Publicado em 10/08/2021

Prazo de entrega recomendado: ***12/08/2021***

Você deve aprender a utilizar um terminal de comandos, realizar operações básicas no controle de versão Git, compilar e testar um programa em C e submeter uma tarefa na disciplina. Esta tarefa serve para testar o sistema de submissão e não fará parte da média de notas.

Linha de comandos

Para fazer as tarefas que são solicitadas nesta disciplina, você deve se familiarizar com a interação com o computador através de linha de comando. No GNU/Linux, um dos interpretadores de comando mais utilizado é o Bash. Nessa tarefa, iremos ver como dar comandos básicos ao computador utilizando o Bash. Caso queira, você poderá utilizar outros interpretadores de comando.

Se você não tem uma distribuição GNU/Linux instalada em seu computador, consulte aqui como instalar o Ubuntu. Você também pode instalar o Ubuntu (ou outras distribuições GNU/Linux) diretamente no Windows usando WSL. Para saber mais detalhes sobre o processo de instalação, ou sempre que tiver dúvidas, consulte os tutoriais feitos especialmente para esta disciplina.

Para começar a nos ambientar com o terminal, vamos criar um diretório e ver o seu conteúdo. Para isso, abra o terminal e digite `mkdir diretorio`. Seu terminal deve se parecer como o seguinte.

```
user@host:~$ mkdir diretorio
```

Para navegar para esse diretório, podemos utilizar o comando `cd`.

```
user@host:~$ cd diretorio
```

Agora, liste o conteúdo desse diretório.

```
user@host:~/diretorio$ ls
user@host:~/diretorio$ ..
```

O nosso diretório contém apenas os diretórios padrões `.` e `..`. O primeiro corresponde ao diretório atual e o segundo ao diretório pai. Esses arquivos (e quaisquer outros começando com `.`) não são mostrados por padrão. Para ver todos os arquivos, use a opção `-a` e para ver detalhes sobre os arquivos, a opção `-l`.

```
user@host:~/diretorio$ ls -la
drwxrwxr-x 2 user user 4096 ago 28 14:32 .
drwxrwxr-x 5 user user 4096 ago 28 14:28 ..
```

Para aprendermos os comandos para copiar e mover arquivos, crie um arquivo nesse diretório utilizando o comando `touch`. Será criado um arquivo sem conteúdo.

```
user@host:~/diretorio$ touch arquivo.txt
```

Se você listar o conteúdo desse diretório, deverá ver o arquivo que acabou de criar.

Agora, crie um subdiretório.

```
user@host:~/diretorio$ mkdir sub-dir
user@host:~/diretorio$ ls -l
-rw-rw-r-- 1 user user    0 ago 28 14:33 arquivo.txt
drwxrwxr-x 2 user user 4096 ago 28 14:33 sub-dir
```

Copie o arquivo `arquivo.txt` para esse subdiretório com o comando `cp`.

```
user@host:~/diretorio$ cp arquivo.txt sub-dir/
```

Crie um novo arquivo.

```
user@host:~/diretorio$ touch arquivo2.txt
```

Mova esse novo arquivo para o subdiretório com o comando `mv`.

```
user@host:~/diretorio$ mv arquivo2.txt sub-dir/
```

Navegue para o subdiretório e liste o seu conteúdo.

```
user@host:~/diretorio$ cd sub-dir
user@host:~/diretorio/sub-dir$ ls
arquivo2.txt  arquivo.txt
```

Para remover um arquivo, utilize o comando `rm`.

```
user@host:~/diretorio/sub-dir$ rm arquivo.txt
```

Volte para o diretório pai e remova o subdiretório com **todo** o seu conteúdo.

```
user@host:~/diretorio/sub-dir$ cd ..
user@host:~/diretorio$ rm -r sub-dir
```

Instalando o compilador

Para compilar programas escritos em C no seu computador, é necessário ter uma série de programas responsáveis por traduzir os vários códigos-fontes em arquivos objetos com as instruções de linguagem de máquina e depois juntá-los (*link*) em um arquivo executável pelo sistema operacional. O conjunto desses programas é chamado de *toolchain*.

Além disso, as tarefas dessa disciplina virão acompanhadas de um arquivo escrito em Python chamado `testar.py` que ajuda a compilar e testar cada uma das tarefas. Não é necessário utilizar esse arquivo para realizar as tarefas, mas testar e aprender a criar testes automatizados evita que você gaste um tempo enorme testando seus programas de maneira manual.

No Ubuntu, ou em distribuições GNU/Linux baseadas em Debian, você pode instalar o toolchain padrão e um interpretador Python 3.x com os comandos abaixo.

```
user@host:~$ sudo apt update
user@host:~$ sudo apt install build-essential python3
```

Execute

```
user@host:~$ gcc --version
user@host:~$ python3 --version
```

pare ter certeza de que a instalação funcionou e verificar a versão dos programas instalados. Para instalar Python em outras distribuições GNU/Linux, você pode consultar o site oficial do Python, ou para o Windows, você pode seguir esse tutorial.

Usando o Git

Nesta disciplina, usaremos so Git para desenvolver e submeter as tarefas de programação. Git é uma ferramenta de controle de versão distribuída, que serve para manter o histórico de arquivos de maneira organizada e facilmente compartilhada por várias pessoas. Não é objetivo dessa disciplina nos aprofundarmos nessa ferramenta, mas é necessário saber realizar comandos básicos e utilizar o repositório efetivamente.

Antes de começar, é preciso instalar o Git para torná-lo disponível em seu computador. No Ubuntu, abra o terminal e execute o comando abaixo.

```
user@host:~$ sudo apt install git
```

Verifique se a instalação ocorreu com sucesso com o comando `git --version`.

Se você ainda não tem e não conhece o Git, você pode ler o seguinte tutorial de introdução, que explica como instalar o software em diferentes sistemas operacionais além dos comandos básicos.

No contexto da disciplina, o Git é uma ferramenta para o acompanhamento das tarefas feitas durante o semestre. A partir dele, @s estudantes terão acesso a uma série de tarefas que servirão de avaliação.

Clonando seu repositório pessoal

Nos primeiros dias, o professor irá criar um repositório no Gitlab do Instituto de Computação (IC) para cada estudante. Cada estudante do curso 34 ou 42 tem uma conta criada automaticamente nessa plataforma. Antes de ser possível criar o repositório, é necessário que cada estudante entre em https://gitlab.ic.unicamp.br e realize o login pela primeira vez utilizando a mesma senha do ambiente Linux. Se você não é desses cursos, o seu professor irá fornecer instruções separadamente sobre como obter sua senha.

Uma vez criado o repositório, é possível cloná-lo para a sua máquina, realizar alterações e depois sincronizá-lo com o repositório remoto. O endereço do seu repositório será

https://gitlab.ic.unicamp.br/mc202-2021/raXXXXXX.git

substituindo-se `XXXXXX` por seu número de RA. Você pode conferir entrando no link. Na página principal do seu repositório, clique no botão `Clone` e copie a URL sobre `Clone with HTTPS`. Abra o terminal dentro do diretório em que ficará guardado o repositório, digite `git clone` e em seguida cole a URL copiada:

```
user@host:~$ git clone https://gitlab.ic.unicamp.br/mc202-2021/raXXXXXX.git
Cloning into 'raXXXXXX'...
Username for 'https://gitlab.ic.unicamp.br': raXXXXXX
Password for 'https://raXXXXXX@gitlab.ic.unicamp.br':
```

Pressione **enter** e realize a autenticação com o mesmo nome de usuário e senha que utilizou para acessar https://gitlab.ic.unicamp.br.

Dentro da pasta que você acabou de criar, deve haver um diretório `tarefa00`. Se não, espere alguns minutos e digite `git pull` dentro do pasta `raXXXXXX` recém criada.

Submissão de tarefas

Durante o semestre, serão realizadas diversas tarefas. A maioria delas consiste em escrever um ou mais programas em C que resolvem determinados problema. Vamos ver um exemplo:

Escreva um programa chamado `bomdia.c` que lê um nome no terminal e dá bom dia.

Entrada

A entrada do programa é o que o usuário irá digitar quando executa o programa. Nesse exemplo, é apenas um nome de uma pessoa.

```
José
```

Saída

A saída o programa é o que o programa irá imprimir na tela após o usuário digitar a entrada. Nesse exemplo, para a entrada acima queremos que o programa tenha a saída:

```
Bom dia, José.
```

Resolvendo o exercício

Para resolver essa tarefa, precisamos criar um arquivo em C faz o solicitado. Para isso, você pode utilizar qualquer editor de texto. Uma sugestão é utilizar os VS Code. Vamos abrir o editor do texto e criar um arquivo `bomdia.c` dentro da pasta `raXXXXXX/tarefa00`.

Em tarefas futuras, você deverá criar todo o programa. Nesse primeiro dia, vamos apenas copiar código abaixo e salvar um arquivo `bomdia.c`.

```c
#include <stdio.h>

int main() {
    char nome[30];
    scanf("%s", nome);
    printf("Bom dia, %s.\n", nome);
}
```

Testando o exercício

Vamos verificar se o arquivo realmente foi criado no diretório correto. Vá até a linha de comandos e digite

```
user@host:~$ cd raXXXXXX/tarefa00
user@host:~/raXXXXXX/tarefa00$ ls -l
total 16
-rw-rw-r-- 1 user user   113 ago 28 14:48 bomdia.c
-rw-rw-r-- 1 user user  6209 ago 28 14:47 testar.py
drwxrwxr-x 2 user user  4096 ago 28 14:47 testes
```

Antes de executar o programa que você criou, é necessário compilá-lo.

```
user@host:~/raXXXXXX/tarefa00$ gcc -std=c99 -Wall -Werror bomdia.c -o bomdia
```

Isso irá compilar o arquivo `bomdia.c` em um arquivo executável `bomdia`. Podemos verificar isso listando o diretório. Depois, executamos o programa recém criado.

```
user@host:~/raXXXXXX/tarefa00$ ls -l
-rwxrwxr-x 1 user user 16792 ago 28 14:48 bomdia
-rw-rw-r-- 1 user user   113 ago 28 14:48 bomdia.c
-rw-rw-r-- 1 user user  6209 ago 28 14:47 testar.py
drwxrwxr-x 2 user user  4096 ago 28 14:47 testes
user@host:~/raXXXXXX/tarefa00$ ./bomdia
```

O console vai ficar piscando esperando que você digite algo. Digite seu nome e aperte Enter.

```
user@host:~/raXXXXXX/tarefa00$ ./bomdia
Simone de Beauvoir
Bom dia, Simone.
```

Deve ter aparecido uma mensagem de bom dia! Se não, verifique se não fez todos os passos acima.

Você acabou de testar manualmente o seu programa. No mundo real, é importante criar testes automatizados. Na nossa disciplina, em cada tarefa, haverá um arquivo `testar.py` que serve para você testar a sua tarefa localmente. Esse arquivo é incluído no diretório da tarefa. Para testar usando o script, abra o terminal dentro do diretório que contém o arquivo Python e digite

```
user@host:~/raXXXXXX/tarefa00$ python3 testar.py

Executando teste_1_bomdia...
  -> testes/bomdia.in passou
teste_1_bomdia: OK

...
```

Ele deve dar uma mensagem falando que um teste correspondente ao primeiro exercício `bomdia.c` passou. Esse é um bom momento para você guardar as suas alterações!

Registrando alterações

As alterações realizadas anteriormente foram no diretório de trabalho. Agora é necessário **registrá-las** no repositório local. No jargão git, queremos realizar um commit, que serve para documentar o desenvolvimento de sua tarefa. Primeiro, vamos revisar as modificações que acabamos de realizar.

```
user@host:~/raXXXXXX/tarefa00$ git status
No ramo master
Your branch is up to date with 'origin/master'.

Arquivos não monitorados:
  (utilize "git add <arquivo>..." para incluir o que será submetido)
    bomdia.c

nada adicionado ao envio mas arquivos não registrados
estão presentes (use "git add" to registrar)
```

A mensagem acima explica que há um arquivo novo `bomdia.c` que ainda não está sendo monitorado pelo repositório de trabalho. No terminal, digite `git add .` para adicionar todos as alterações no diretório de trabalho à **área de preparação** (*staging*). Isso irá adicionar qualquer arquivo que estiver no diretório atual, por isso, sempre verifique o status antes e depois.

```
user@host:~/raXXXXXX/tarefa00$ git add .
user@host:~/raXXXXXX/tarefa00$ git status
No ramo master
Your branch is up to date with 'origin/master'.

Mudanças a serem submetidas:
  (use "git restore --staged <file>..." to unstage)
    new file:   bomdia.c
user@host:~/raXXXXXX/tarefa00$ git diff
```

Agora podemos registrar essas alterações. Antes de realizar o primeiro `commit` em uma máquina com uma instalação nova do git, é necessário definir seu endereço de e-mail e nome de usuário conforme o exemplo a seguir. O endereço de e-mail é utilizado para identificar quais alterações foram realizadas por cada autor. Por isso, certifique-se de usar sempre o mesmo endereço de e-mail. Nesta disciplina, recomendamos que você utilize o seu email institucional, como no exemplo.

```
user@host:~$ git config --global user.email "XXXXXX@dac.unicamp.br"
user@host:~$ git config --global user.name "Fulana da Silva"
```

O próximo passo é confirmar as alterações, ou seja, fazer um commit. Será necessário adicionar uma mensagem de alterações, o que pode ser feito adicionando o texto entre aspas após o comando commit com a flag `-m`. Certifique-se de incluir uma mensagem que realmente seja descritiva das alterações realizadas nesse commit.

```
user@host:~/raXXXXXX/tarefa00$ git commit -m "cria exercício bomdia.c"
[master f91b22d] cria exercício bomdia.c
 1 file changed, 7 insertions(+)
 create mode 100644 tarefa00/bomdia.c
```

Sincronizando repositórios

Tudo que você fez até agora está registrado apenas no seu repositório local. As alterações ainda **não** foram enviadas para o repositório remoto. Por último, é necessário executar o comando `git push` no terminal e realizar a autenticação.

```
user@host:~/raXXXXXX/tarefa00$ git push
Username for 'https://gitlab.ic.unicamp.br': raXXXXXX
Password for 'https://raXXXXXX@gitlab.ic.unicamp.br':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 457 bytes | 457.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To https://gitlab.ic.unicamp.br/mc202-2020/raXXXXXX.git
   6018803..f91b22d  master -> master
```

Verificando o resultado de suas tarefas

Vocês devem ter recebido um e-mail com uma senha para acessar o site https://mc202.ic.unicamp.br. Essa site registra todas as correções recebidas no repositório remoto, feedback dos monitores e relatórios de erros. Tome cuidado com sua senha, ela não pode ser alterada. Por isso, não perca nem compartilhe sua senha com ninguém. Se você não recebeu a senha no seu **e-mail institucional**, então converse com o professor ou monitor.

Boa noite

Quando você executou o programa `testar.py` ou viu o site de notas, deve ter percebido que algum teste falhou. Isso é porque pode haver mais de um exercício por tarefa. Para concluir `tarefa00`, faça um programa `boanoite.c` que lê vários nomes e dá boa noite, conforme entrada e saída a seguir.

Entrada

```
3
José
Maria
Manuela
```

Saída

```
Boa noite, José.
Boa noite, Maria.
Boa noite, Manuela.
```

Crie um arquivo `boanoite.c` conforme o modelo abaixo. Depois compile usando o `gcc` e teste manualmente seu programa.

```c
#include <stdio.h>

int main() {
    char nome[30];
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s", nome);
        printf("Boa noite, %s.\n", nome);
    }
}
```

Depois de testado manualmente, vamos tentar com o script de teste automático. Para testar apenas o teste de boa noite, passamos um parâmetro para o arquivo `testar.py`, especificando o nome da unidade de teste escolhida.

```
user@host:~/raXXXXXX/tarefa00$ python3 testar.py boanoite

Executando teste_2_boanoite...
teste_2_boanoite: FALHOU
  -> você deve criar um arquivo Makefile
```

Ele está reclamando de que não há um arquivo `Makefile`. Para esse exercício, vamos exemplificar também como criar um arquivo `Makefile`. Esse tipo de arquivo serve para guardar um conjunto de regras para compilar seu programa automáticamente, ao invés de ter que digitar o comando de compilação manualmente toda vez. Para isso, abra seu editor e crie um arquivo `Makefile` conforme o modelo abaixo. É muito importante que esse arquivo seja indentado usando **tabs** ao invés de espaços.

```makefile
CFLAGS = -std=c99 -Wall -Werror -Werror=vla -g
LDFLAGS = -lm

.PHONY: all clean

all: bomdia boanoite

bomdia: bomdia.c
	gcc $(CFLAGS) bomdia.c -o bomdia $(LDFLAGS)

boanoite: boanoite.c
	gcc $(CFLAGS) boanoite.c -o boanoite $(LDFLAGS)

clean:
	rm -f bomdia boanoite
```

Agora, para compilar todos os programas, basta digitar `make`.

```
user@host:~/raXXXXXX/tarefa00$ make
gcc -std=c99 -Wall -Werror -Werror=vla -g bomdia.c -o bomdia -lm
gcc -std=c99 -Wall -Werror -Werror=vla -g boanoite.c -o boanoite -lm
```

Pronto, agora você já pode testar, registrar e enviar suas alterações para o repositório remoto.

Voltar Próximo →