MAC 216 - EP 2

Técnicas de Programação 1 – simplemail
Script em Bash

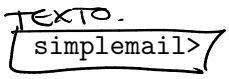
Data de entrega: 9/10/2021 (sem extensões!)

Introdução

Neste exercício, você deve implementar o programa `simplemail`, que é um sistema simples de e-mail local. O programa deve permitir que diferentes usuários[1] em uma mesma máquina possam trocar mensagens assíncronas entre si, ou seja, um usuário deve poder consultar as mensagens enviadas para ele mesmo quando ele não estava logado. O sistema só precisa dar suporte a mensagens de texto puro e deve ser escrito inteiramente em `bash`, num único arquivo chamado `mac216-simplemail.sh`.

1 Funcionamento

Ao ser executado, o programa deve apresentar um *prompt* para o usuário inserir comandos:



Os comandos válidos são:

- `createuser username password` Cria um novo usuário

  Se o usuário já existir, o comando não deve fazer nada a não ser apresentar uma mensagem de erro.

- `passwd username oldpassword newpassword` Modifica a senha de um usuário existente

  Se a senha original não corresponder ao que foi registrado anteriormente ou se o usuário não existir, o comando não deve fazer nada a não ser apresentar uma mensagem de erro.

- `login username password` Faz o *login* do usuário

  Se o usuário não existir ou se a senha estiver incorreta, o comando não deve fazer nada a não ser apresentar uma mensagem de erro.

- `listusers` Lista os usuários cadastrados no programa

- `msg user` Envia uma mensagem para o usuário `user`

  Se o usuário não existir, o comando não deve fazer nada a não ser apresentar uma mensagem de erro.

- `list` Mostra a lista de mensagens recebidas

[1] Na verdade, usuários fictícios, que existem apenas dentro do próprio programa; não é preciso se preocupar com diferentes usuários Unix/Linux.

- `read msgnum` Mostra o conteúdo da mensagem número `msgnum`

  Se a mensagem não tiver sido lida anteriormente, seu *status* de mensagem não-lida deve ser alterado.

- `unread msgnum` Redefine a mensagem número `msgnum` como não-lida
- `delete msgnum` Apaga a mensagem `msgnum`
- `quit` Finaliza o programa

Os comandos `listusers`, `msg`, `list`, `read`, `unread` e `delete` não devem fazer nada a não ser apresentar uma mensagem de erro caso o usuário ainda não tenha feito *login*.

1.1 Interface com o usuário

A lista de mensagens deve seguir um formato similar a este (você pode usar outro formato de data desde que ele seja facilmente compreensível por humanos, como "20/09/2021 02:17 PM" ou "jul 8 14:33"):

```
simplemail> list
1 |   | seg 20 set 2021 16:01:22 -03 | turing
2 | N | seg 20 set 2021 16:02:58 -03 | lovelace
simplemail>
```

A primeira coluna indica o número da mensagem; a segunda indica (com a letra N) se a mensagem é nova (ainda não foi lida) ou não; a terceira indica a data de envio; e a quarta, o remetente.

As mensagens devem ser apresentadas ao usuário da seguinte forma:

```
simplemail> read 1
De: turing
Uma máquina de Turing é capaz de simular um analytical
engine; será que o inverso também é verdadeiro?
simplemail>
```

2 Detalhes de implementação

Todos os arquivos gerados pelo programa (como a lista de usuários cadastrados, as mensagens enviadas e recebidas etc.) deverão ser armazenados dentro do diretório `simplemail` no diretório corrente. O programa deve criar esse diretório e os demais arquivos e subdiretórios necessários automaticamente. Se o usuário sair do programa e voltar, todos os usuários e mensagens criados anteriormente devem continuar existindo, mas se esse diretório for apagado, na próxima execução do programa ele deve começar "do zero".

Os diferentes usuários criados pelo programa existem apenas dentro dele e não têm nenhuma relação com os usuários Unix/Linux cadastrados na máquina em que o programa é executado. Ou seja, você não precisa se preocupar com as permissões dos arquivos etc. para que usuários diferentes possam acessar o sistema: Apenas um usuário Unix/Linux vai executar o programa.

3 Algumas dicas

Desde que seu programa seja feito usando técnicas de script com `bash`, você pode escrevê-lo como quiser, mas aqui vão algumas dicas que podem ser úteis:

- Se você quiser executar um comando e ignorar quaisquer mensagens de erro, basta fazer `comando 2>/dev/null`
- Quando se está digitando no terminal, é possível indicar "fim" com o caracter `CTRL-D` ("fim de arquivo"); experimente fazer `cat > algum-arquivo`, digitar alguma coisa e terminar com `CTRL-D`
- Quando você faz algo como `for arquivo in *; do...`, você provavelmente espera que, se não houver nenhum arquivo, o laço `for` não seja executado. **Não** é isso que acontece! Nesse caso, ao invés de substituir o `*` pela lista de arquivos, como não há arquivo nenhum a substituição não acontece e o laço `for` é executado uma vez com a variável `$arquivo` contendo o caracter `*`. O mesmo problema acontece se você fizer, por exemplo, `ls *` em um diretório vazio. Então, para obter uma lista de arquivos que pode ou não estar vazia, utilize `find diretorio-a-procurar outro-diretorio-a-procurar -type f`. Esse comando lista todos os arquivos, mesmo dentro de subdiretórios, com o caminho completo (`diretorio-a-procurar/outro-diretorio/nome-do-arquivo`). Outra opção é fazer apenas `ls diretorio-a-procurar`; nesse caso, a lista não inclui o caminho completo (apenas `nome-do-arquivo`).
- Sempre que você for comparar *strings* no shell, coloque-as entre aspas. Se você não fizer isso, uma *string* vazia pode causar erros de sintaxe. Por exemplo, `if [ ola = $var ]; then...` vai falhar se `$var` for vazia, porque o shell vai processar `if [ ola = ]; then...`, o que não faz sentido. Isso vale também quando a *string* é o resultado de um comando, como `if [ "$(grep...)" = blah ]; then...`.

Além dos comandos básicos usados em scripts que você já conhece (`while`, `if`, `cp`, `ls`, `grep`, `cut` etc.), você pode querer consultar a documentação dos comandos abaixo (você não é obrigado a usá-los, mas eles provavelmente vão facilitar seu trabalho):

- `date +%s` e seu reverso, `date --date="jan 1 1970 UTC + XXX seconds"`
- A opção `-p` do comando `mkdir`
- `sed -e "s/blah/bleh/g"`

Finalmente, **não se esqueça de prestar atenção nas aspas**: às vezes elas são necessárias, às vezes elas não devem ser usadas e às vezes elas não fazem diferença. Se você encontrar algum comportamento inesperado no seu script, comece conferindo as aspas!

4 Bônus

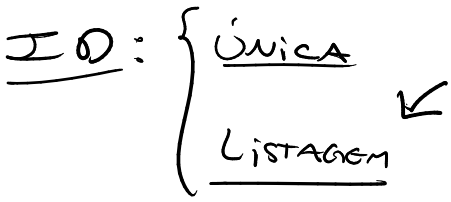
Se você quiser ganhar um ponto extra neste exercício, há duas melhorias que você pode implementar:

- Modifique os comandos `msg`, `list` e `read` para que as mensagens possuam um assunto (uma linha) e um corpo (que pode ter múltiplas linhas):

```
simplemail> login lovelace analyticalengine
simplemail> read 1
De: turing
Assunto: Analytical Engine vs Turing Machine
```

```
Uma máquina de Turing é capaz de simular um analytical
engine; será que o inverso também é verdadeiro?
simplemail> msg lovelace
Qual o assunto? Alunos sortudos
Qual a mensagem? Termine com "CTRL-D"
Os alunos atuais têm sorte, podem fazer seus
programas em bash ao invés de programar
tudo em linguagem de máquina!
simplemail> list

2 | N | seg 20 set 2021 16:02:58 -03 | lovelace | Alunos sortudos
simplemail>
```



Os comandos `head -n num`, `tail -n num` e `tail -n +num` podem ser úteis para o processamento do assunto.

- Faça a lista de mensagens sempre aparecer ordenada pela data de envio. Para isso, aprenda sobre o comando `sort`, em particular as opções `-n`, `-t` e `-k`. Lembre-se que `sort` precisa que os itens da lista a ser ordenada estejam cada um em uma linha separada!

5 Exemplo de uso

Na interação com o usuário, o programa deve se comportar mais ou menos assim:

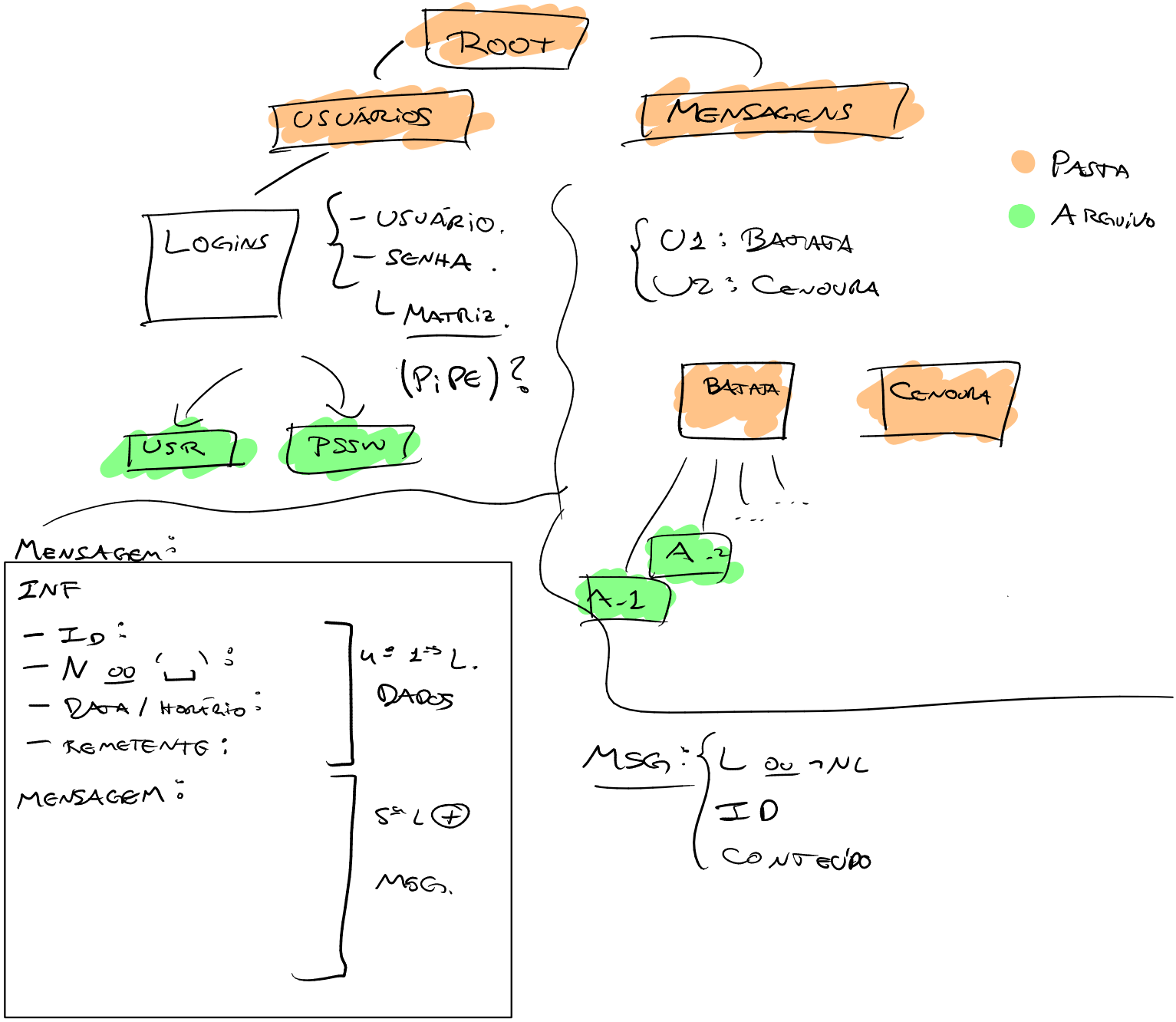
```
simplemail> createuser lovelace analyticalengine
simplemail> createuser turing amachine
simplemail> login turing amachine
simplemail> msg lovelace
Qual a mensagem? Termine com "CTRL-D"
Uma máquina de Turing é capaz de simular um analytical
engine; será que o inverso também é verdadeiro?
simplemail> login lovelace analyticalengine
simplemail> list
1 | N | seg 20 set 2021 16:01:22 -03 | turing
simplemail> read 1
De: turing
Uma máquina de Turing é capaz de simular um analytical
engine; será que o inverso também é verdadeiro?
simplemail> list
1 |   | seg 20 set 2021 16:01:22 -03 | turing
simplemail> delete 1
simplemail> list
simplemail> msg turing
Qual a mensagem? Termine com "CTRL-D"
Os alunos atuais têm sorte, podem fazer seus
programas em bash ao invés de programar
tudo em linguagem de máquina!
simplemail> login turing amachine
simplemail> list
1 | N | seg 20 set 2021 16:02:58 -03 | lovelace
simplemail> quit
OK, até mais!
```

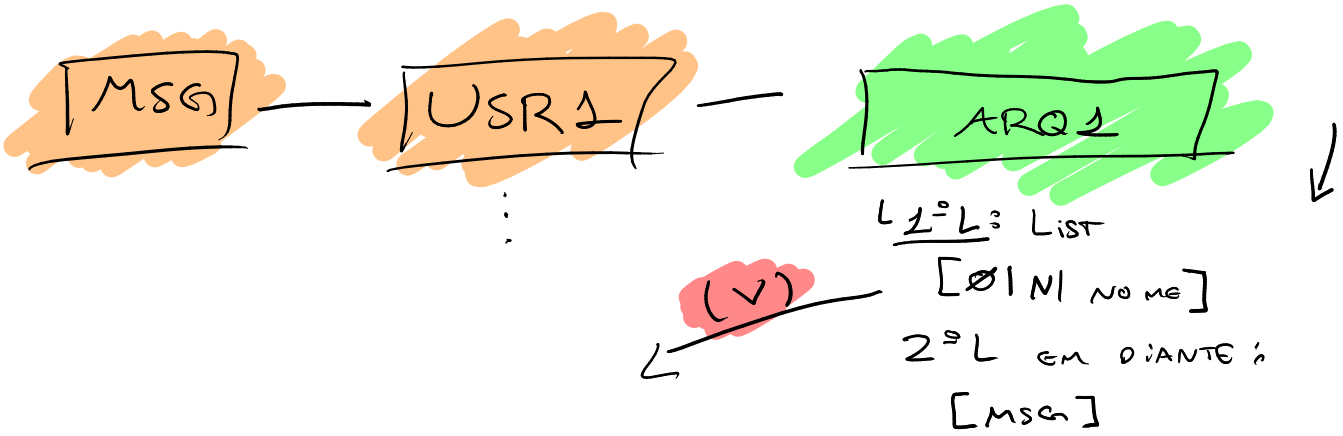
6 Recomendações Finais

Seu código deve ser **elegante, claro e eficiente**. Os nomes das variáveis e funções devem ser o mais claros possíveis. Não misture línguas (faça tudo em português ou tudo em inglês, sem misturar as duas).

O trabalho deve ser realizado em grupos de 1 ou 2 alunos. A entrega será feita por meio do eDisciplinas. Caso seja feito em dupla, apenas um dos membros da dupla deve submeter o exercício no edisciplinas, mas o arquivo deve conter claramente o nome e NUSP de ambos os alunos. No caso de dupla, ambos os alunos devem obrigatoriamente participar ativamente da escrita do programa; vocês podem optar por fazer programação em par ou dividir o código de forma que cada um escreva uma parte e revise a outra parte.

Obviamente, *o plágio é terminantemente proibido*. Caso seja detectado plágio, todos os membros do grupo plagiador e todos os membros do grupo plagiado receberão nota -10 no exercício. Portanto, compartilhe seu código apenas com seu colega de grupo.





1 ARQ / MSG:

- ORGANIZAÇÃO.
- LIST.

1°) LIST:

ITERAR PELOS ARQUIVOS.
(ORDEM CRESCENTE INDEX)
PRINTA INF DAS 4° 1°S LINHAS

2°) ENVIAR:

CRIA ARQUIVO.
└ INSERE INF E MSG.

3°) LER: DADO INDEX = i

BUSCA MENSAGEM i)

COMANDOS:

1º) CREATEUSER u USR-NAME PSSWD

└ NO ARQ. "USUÁRIOS":

a) CHECA SE ∄u → CRIA LINHA (u)

b) ∃u → MSG(ERRO)

└ NO ARQ "SENHAS": ↔ 1)a)

a) CRIA LINHA (p)

2º) PASSWD u op np

└ NA PASTA "USR"

a) ARQ "USR": LINHA DE u

└ b) ARQ "SENHAS": LINHA u.

I) op FOR IGUAL AO VALOR ARMAZENADO, TROCA POR np

II) SENÃO MSG(ERRO).

3º) LOGIN u p

└ NA PASTA "USUARIOS":

a) No ARQ. "USUARIOS.USR":

I) ∃u? SIM COLETA ÍNDICE DA LINHA DE u
NÃO MSG(ERRO)

b) No ARQ. "SENHAS. PSSWD"

I) p = u(p)? SIM PARABÉNS!
NÃO MSG(ERRO)

4º) LISTUSERS:

a) NA PASTA "USUARIOS":

CAT (USUARIOS.USR).

5º) MSG USR — SENÃO: MSG(ERRO).

└ SE ∃ USR:

> INSERE MENSAGEM.

└ PASTA "MENSAGENS/USR/"

└ ID DA MENSAGEM: 1+(LS "MENSAGENS/USR/" | WC -L)

└ "MENSAGEM_ID.MSG":

4ºs └ [- ID \n
- N \n
- DATA (DATE) \n
- USR \n
\n] ECHO (————) > MENSAGEM_ID.MSG

5ºs ↓ [

6º) LIST:

FOR (ARQUIVO EM "../MENSAGENS/CURRENT-USER/"):

{ PEGA AS 4ºs LINHAS E PRINTA NO FORMATO.

7º) READ MSGNUM (SE ID VÁLIDO)

PEGA DE ("../MENSAGENS/CURRENT-USER/MENSAGEM_MSGNUM")

DA 5ª LINHA P/ FRENTE.

8º) UNREAD MSGNUM (SE ID VÁLIDO)

DE ("../MENSAGENS/CURRENT-USER/MENSAGEM_MSGNUM"),

MUDA-SE A SEGUNDA LINHA DE '␣' PARA 'N'.

9º) DELETE MSGNUM (SE ID VÁLIDO)

DELETA-SE (RM)("../MENSAGENS/CURRENT-USER/MENSAGEM_MSGNUM").

10º) QUIT: NADA! :)