

# GIT, GDB e testes

LA1/LI2

15 de Março de 2015

# Parte I

## GIT

- 1 Entrar na máquina por ssh
- 2 Meter a password
- 3 Criar a diretoria para guardar o repositório
- 4 Criar o repositório local
- 5 Enviar a primeira cópia para o repositório

#### Exemplo dos passos 1 a 3

```
ssh li2g67.2015@git.alunos.di.uminho.pt  
mkdir li2  
cd li2  
git init --bare  
exit
```



- 1 Criar o repositório local vazio

## Exemplo do passo 4 do slide anterior

```
mkdir li2  
git init
```

- 2 Criar ficheiros na diretoria usando o editor de texto preferido
- 3 Adicionar os ficheiros ao repositório e sincronizar com o repositório remoto

## Exemplo do passo 5 do slide anterior

```
git add *.c  
git commit -m "Commit inicial do repositorio"  
git push li2g67.2015@git.alunos.di.uminho.pt
```

- 1 Fazer um clone do repositório
- 2 Modificar o que se quiser
- 3 Fazer o commit
- 4 Sincronizar com o repositório

### Exemplo

```
git clone li2g67.2015@git.alunos.di.uminho.pt
gedit interp.c
commit -a -m "modifiquei o interp.c para ..."
git push li2g67.2015@git.alunos.di.uminho.pt
```

## Parte II

# Debugger

Para se compilar em modo debug é necessário colocar a opção `-g`.

### Exemplo

```
gcc -Wall -Wextra -ansi -pedantic -O2 -g *.c -o bn
```

**p expressão** Imprime a expressão

**b linha** Coloca um breakpoint numa linha no ficheiro actual

**b ficheiro:linha** Coloca um breakpoint numa linha do ficheiro

**b nome\_de\_função** Coloca um breakpoint numa função

**run** Corre o programa desde o início

**c ou cont** Continua a execução do programa

**s ou step** Executa uma linha

**n ou next** Executa uma linha mas não entra em funções auxiliares

**where** Mostra o stack trace (útil se o programa rebentou por exemplo)

**l ou list** lista o código perto da posição actual onde está a ser executado

**up** subir na stack

**down** descer na stack





```
#include <stdio.h>
int g(char *s, int n) {
    int j;

    for(j = 0; j < n; j++)
        s[j] += 'A' - 'a';
    return 0;
}
int f(char *s) {
    return g(s, 100);
}
int main() {
    char *buf = NULL;

    f(buf);
    printf("%s\n", buf);
}
```



Se o programa crashou (e.g. segmentation fault) o mais fácil é fazer o seguinte:

- 1 Compilar o programa como mostrado acima
- 2 Antes de correr o programa escrever no prompt `ulimit -c unlimited` (pode-se adicionar esta linha ao `/.bashrc` para não termos que fazer isto de cada vez)
- 3 Correr o programa e fazer com que ele estoure
- 4 Escrever `gdb exec core` em que `exec` é o nome do executável
- 5 Dentro do debugger começar por escrever `where` para ver o que aconteceu e usar o comando `up` para navegar na stack e o comando `p` para ver o valor das variáveis

```
rui@omega:/tmp/l$ gcc -ggdb c.c
rui@omega:/tmp/l$ ./a.out
Segmentation fault (core dumped)
rui@omega:/tmp/l$ gdb a.out core
GNU gdb (Ubuntu/Linaro 7.4-2012.04-0ubuntu2.1) 7.4-2012.04
Copyright (C) 2012 Free Software Foundation, Inc.
...
Core was generated by './a.out'.
Program terminated with signal 11, Segmentation fault.
#0  0x080483ef in g (s=0x0, n=100) at c.c:7
7          s[j] += 'A' - 'a';
(gdb) where
#0  0x080483ef in g (s=0x0, n=100) at c.c:7
#1  0x08048423 in f (s=0x0) at c.c:12
#2  0x08048442 in main () at c.c:18
(gdb)
```

```
rui@omega:/tmp/l$ gdb a.out
...
Reading symbols from /tmp/l/a.out...done.
(gdb) run
Starting program: /tmp/l/a.out

Program received signal SIGSEGV, Segmentation fault.
0x080483ef in g (s=0x0, n=100) at c.c:7
7          s[j] += 'A' - 'a';
(gdb) where
#0  0x080483ef in g (s=0x0, n=100) at c.c:7
#1  0x08048423 in f (s=0x0) at c.c:12
#2  0x08048442 in main () at c.c:18
(gdb) p s
$1 = 0x0
```

```
(gdb) up
#1  0x08048423 in f (s=0x0) at c.c:12
12          return g(s, 100);
(gdb) up
#2  0x08048442 in main () at c.c:18
18          f(buf);
(gdb) p buf
$2 = 0x0
(gdb)
```

## Parte III

# Metodologia de testes



- 1 Para cada exemplo, criar dois ficheiros, um com o input e outro com o output, sugere-se que os dois ficheiros tenham o mesmo nome mas extensões diferentes por exemplo `in` e `out`
- 2 Invocar o programa redirecionando o input
- 3 Comparar o output com o esperado



O comando `diff` compara o output obtido com o esperado. Caso sejam iguais, não imprime nada. Caso contrário, imprime a diferença.

```
bn < t01.in | diff - t01.out
```





- 1 Criar uma script com os testes (e.g. `testes.sh`)
- 2 Correr a script utilizando  
`bash testes.sh`



```
for teste in t01 t02 t03
do
echo $teste
bn < $teste.in | diff - $teste.out
done
```