

Linux gdb与makefile使用

4. 掌握简单gdb使用于调试
5. 掌握简单的Makefile编写，了解其运行思想
6. 编写自己的第一个Linux 程序：进度条
7. 学习 git 命令行的简单操作, 能够将代码上传到 Github 上

Linux调试器-gdb使用 [重要点进行讲解]

1.背景

- 程序的发布方式有两种，debug模式和release模式
- Linux gcc/g++出来的二进制程序，默认是release模式
- 要使用gdb调试，必须在源代码生成二进制程序的时候, 加上 -g 选项 [重要]

2. 开始使用

`gdb binFile` 退出: `ctrl + d` 或 `quit` 调试命令:

- `list / l` 行号: 显示binFile源代码，接着上次的位置往下列，每次列10行。 [重要]
- `list / l` 函数名: 列出某个函数的源代码。
- `r`或`run`: 运行程序。 [重要]
- `n` 或 `next`: 单条执行。 [重要]
- `s`或`step`: 进入函数调用 [重要]
- `break(b)` 行号: 在某一行设置断点 [重要]
- `break` 函数名: 在某个函数开头设置断点
- `info break` : 查看断点信息。
- `finish`: 执行到当前函数返回，然后挺下来等待命令
- `print(p)`: 打印表达式的值，通过表达式可以修改变量的值或者调用函数
- `p` 变量: 打印变量值。 [重要]
- `set var`: 修改变量的值
- `continue(或c)`: 从当前位置开始连续而非单步执行程序 [重要]
- `run(或r)`: 从开始连续而非单步执行程序
- `delete breakpoints`: 删除所有断点
- `delete breakpoints n`: 删除序号为n的断点 [重要]
- `disable breakpoints`: 禁用断点
- `enable breakpoints`: 启用断点
- `info(或i) breakpoints`: 参看当前设置了哪些断点
- `display` 变量名: 跟踪查看一个变量，每次停下来都显示它的值
- `undisplay`: 取消对先前设置的那些变量的跟踪
- `until X`行号: 跳至X行
- `breaktrace(或bt)`: 查看各级函数调用及参数
- `info (i) locals`: 查看当前栈帧局部变量的值
- `quit`: 退出gdb

3. 理解

- 和windows IDE对应例子

Linux项目自动化构建工具-make/Makefile

背景

- 会不会写makefile，从一个侧面说明了一个人是否具备完成大型工程的能力
- 一个工程中的源文件不计数，其按类型、功能、模块分别放在若干个目录中，makefile定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作
- makefile带来的好处就是——“自动化编译”，一旦写好，只需要一个make命令，整个工程完全自动编译，极大的提高了软件开发的效率。
- make是一个命令工具，是一个解释makefile中指令的命令工具，一般来说，大多数的IDE都有这个命令，比如：Delphi的make，Visual C++的nmake，Linux下GNU的make。可见，makefile都成为了一种在工程方面的编译方法。
- make是一条命令，makefile是一个文件，两个搭配使用，完成项目自动化构建。

理解

- 依赖例子

实例代码

C代码

```
#include <stdio.h>

int main()
{
    printf("hello Makefile!\n");
    return 0;
}
```

Makefile文件

```
hello:hello.o
    gcc hello.o -o hello
hello.o:hello.s
    gcc -c hello.s -o hello.o
hello.s:hello.i
    gcc -S hello.i -o hello.s
hello.i:hello.c
    gcc -E hello.c -o hello.i
```

```
.PHONY:clean
clean:
    rm -f hello.i hello.s hello.o hello
```

依赖关系

- 上面的文件 `hello`，它依赖 `hello.o`
- `hello.o`，它依赖 `hello.s` `hello.s`，它依赖 `hello.i`
- `hello.i`，它依赖 `hello.c`

依赖方法

- `gcc hello.* -option hello.*`，就是与之对应的依赖关系

原理

- `make`是如何工作的,在默认的方式下，也就是我们只输入`make`命令。那么，
 1. `make`会在当前目录下找名字叫“`Makefile`”或“`makefile`”的文件。
 2. 如果找到，它会找文件中的第一个目标文件（`target`），在上面的例子中，他会找到“`hello`”这个文件，并把这个文件作为最终的目标文件。
 3. 如果`hello`文件不存在，或是`hello`所依赖的后面的`hello.o`文件的文件修改时间要比`hello`这个文件新（可以用 `touch` 测试），那么，他就会执行后面所定义的命令来生成`hello`这个文件。
 4. 如果`hello`所依赖的`hello.o`文件不存在，那么`make`会在当前文件中找目标为`hello.o`文件的依赖性，如果找到则再根据那一个规则生成`hello.o`文件。（这有点像一个堆栈的过程）
 5. 当然，你的C文件和H文件是存在的啦，于是`make`会生成 `hello.o` 文件，然后再用 `hello.o` 文件声明`make`的终极任务，也就是执行文件`hello`了。
 6. 这就是整个`make`的依赖性，`make`会一层又一层地去找文件的依赖关系，直到最终编译出第一个目标文件。
 7. 在找寻的过程中，如果出现错误，比如最后被依赖的文件找不到，那么`make`就会直接退出，并报错，而对于所定义的命令的错误，或是编译不成功，`make`根本不理。
 8. `make`只管文件的依赖性，即，如果在我找了依赖关系之后，冒号后面的文件还是不在，那么对不起，我就不工作啦。

项目清理

- 工程是需要被清理的
- 像`clean`这种，没有被第一个目标文件直接或间接关联，那么它后面所定义的命令将不会被自动执行，不过，我们可以显示要`make`执行。即命令——“`make clean`”，以此来清除所有的目标文件，以便重编译。
- 但是一般我们这种`clean`的目标文件，我们将它设置为伪目标,用 `.PHONY` 修饰,伪目标的特性是，总是被执行的。
- 可以将我们的 `hello` 目标文件声明成伪目标，测试一下。

Linux第一个小程序 - 进度条

\r&&\n

- 回车概念
- 换行概念
- 老式打字机的例子

行缓冲区概念

什么现象？

```
#include <stdio.h>

int main()
{
    printf("hello Makefile!\n");
    sleep(3);
    return 0;
}
```

什么现象??

```
#include <stdio.h>

int main()
{
    printf("hello Makefile!");
    sleep(3);
    return 0;
}
```

什么现象???

```
#include <stdio.h>

int main()
{
    printf("hello Makefile!");
    fflush(stdout);
    sleep(3);
    return 0;
}
```

进度条代码

```
#include <unistd.h>
#include <string.h>

int main()
{
    int i = 0;
    char bar[102];
    memset(bar, 0 ,sizeof(bar));
    const char *lable="|/-\\";
    while(i <= 100 ){
        printf("[%s-100s] [%d%%] [%c]\r", bar, i, lable[i%4]);
        fflush(stdout);
        bar[i++] = '#';
        usleep(10000);
    }
    printf("\n");
    return 0;
}
```

```
}
```

```
[hb@MiWiFi-R1CL-srv test]$ ./hello
#####
```

```
][17%][ / ]
```

使用 git 命令行

安装 git

```
yum install git
```

在 Github 创建项目

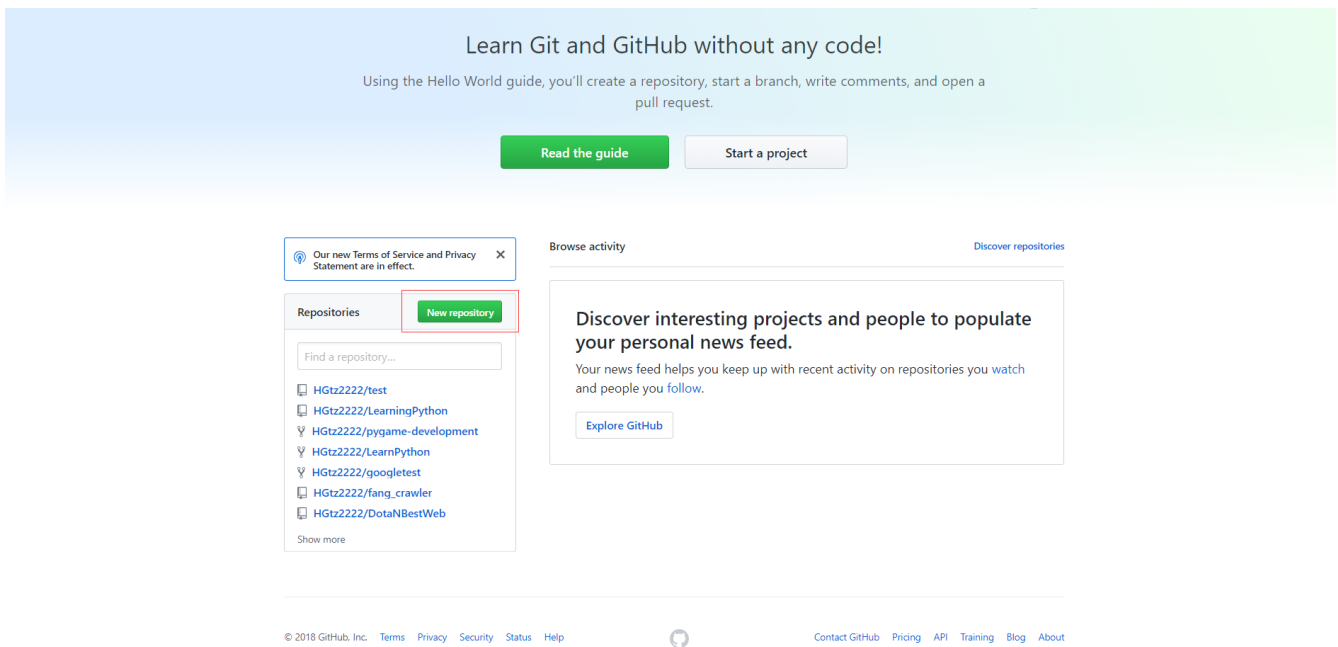
使用 Github 创建项目

注册账号

这个比较简单, 参考着官网提示即可. 需要进行邮箱校验.

创建项目

1. 登陆成功后, 进入个人主页, 点击左下方的 New repository 按钮新建项目




2. 然后跳转到的新页面中输入项目名称(注意, 名称不能重复, 系统会自动校验. 校验过程可能会花费几秒钟). 校验完毕后, 点击下方的 Create repository 按钮确认创建.

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 HGtz2222 ▾

Repository name

Great repository names are short and memorable. Need inspiration? How about **super-duper-eureka**.

Description (optional)



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.



Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾

Add a license: None ▾



Create repository

3. 在创建好的项目页面中复制项目的链接, 以备接下来进行下载.

if you've done this kind of thing before

or **HTTPS** **SSH** `https://github.com/HGtz2222/test2.git`



repository include a [README](#), [LICENSE](#), and [.gitignore](#).

下载项目到本地

创建好一个放置代码的目录.

```
git clone [url]
```

这里的 url 就是刚刚建立好的 项目 的链接.

三板斧第一招: git add

将代码放到刚才下载好的目录中

```
git add [文件名]
```

将需要用 git 管理的文件告知 git

三板斧第二招: git commit

提交改动到本地

```
git commit .
```

最后的 "." 表示当前目录

提交的时候应该注明提交日志, 描述改动的详细内容.

三板斧第三招: git push

同步到远端服务器上

```
git push
```

需要填入用户名密码. 同步成功后, 刷新 Github 页面就能看到代码改动了.

配置免密码提交

<https://blog.csdn.net/camillezj/article/details/55103149>