

C - Simple Shell

وَسْمَى عِنْيَيْنِ الـ Shell ٩٦. الـ Shell هُو بِرَاجِمٍ يَسْتَقْبِلُ أَوْامِرَ مُثُلَّ (ls, pwd, cd) وَيَنْفَذُهَا عَنْ طَرِيقِ استِعْدَادِ بِرَاجِمٍ أُخْرَى فِي المُطَهَّرِ

`$./bin/ls`

هَذَا يَعْنِي نَفْذُ الْبِرَاجِمَ فِي الْمَسَارِ /bin/ls دَأْنَهُ لِلْمُنَفَّعَاتِ

(١)

٩٩. Simple Shell 0.1

شُهُويِّ بِرَاجِمٍ بِسِيْطٍ :

- يَظْهُرُ لَكَ #Cisfun# أَوْدَى Prompt
- يَنْتَظِرُ الْمُسْتَهْدِمُ يَكْتُبُ أَوْمَرَ
- يَنْفَذُ الْأَوْمَرَ بِاستِعْدَادِ Execve
- إِذَا الْمُسْتَهْدِمُ كَتَبَ (EOF) Ctrl+D يَنْفَذُهُ الْبِرَاجِمُ
- تَطْلُبُ رسَالَةً خَطَّاءً إِذَا الْأَوْمَرُ مُوْجَدٌ.

وَمِنْ أَدَمَالِ الْأَوْلَى رَاجِمٌ شَنَدَهَا ٩٩

- Getline() ← تَقْرَأُ أَوْمَرَ الْمُسْتَهْدِمِ
- fork() ← تَنْسَخُ نَسْخَةً جَوِيدَةً مِنَ الْبِرَاجِمِ (عَلَيْهِ بِرِيءَةِ)
- Execve() ← تَنْفِذُ الْبِرَاجِمَ الْجَوِيدَةَ دَاخِلَ الدَّلِيلِ الْبَرِيءِ
- Wait() ← تَسْتَعْتَرُ الْفَلَيْلَةَ تَخَلِّهِ
- Perror() ← تَطْلُبُ رسَالَةً خَطَّاءً إِذَا فَشَلَ تَنْفِذُ الْأَوْمَرِ

(مَلَدَّهَاتِ مَهِمٍ) :-

* لَدَمْ كَلِّ مَلْفِي readme.md يَمْرُ عَلَيْهِ

* لَدَمْ يَكُونُ عَنْكَ مَلْفِي README.md

* الْبِرَاجِمُ مَا يَقْبِلُ أَوْمَرَ مِنْهَا ls -l حَالِيَا (سِرَاجٌ تَضَيِّفُهَا لَدَقَّةً فِي الْمَادِلِ اِنْتَلِي)

(٢)

٩٩. نَرْجُ بِطَرِيقِ ثَانِيَةِ ...

تَحْلِيَّوا بِاِنْكِسَمٍ فِي فُرِيقِ مُجِنِّزٍ وَطَلَبُ مِنْكُمُ الْمَهْنِسِ الْمَسْؤُلِ لِتَسْوِيَنِ بِرَاجِمٍ لِيَقْدِرُ بِسْتَقْبِلُ أَوْمَرَ مِنَ الْمُسْتَهْدِمِ زَيِّ (./bin/z) وَيَنْفَذُهَا

لِشَكْلِ عَلَى كَفَافٍ ٩٦

الْمُسْتَهْدِمُ يَفْتَحُ الْتِرْمَنَالِ وَيَكْتُبُ :

`./hash`

طَبِيبُ دَمَتِ الْمَطَهُوبِ ٩٩

الْمَطَهُوبُ هَذِهِ :

① تَظْهُرُ لَهُ بِرَوْمَبَتْ زَيِّ ٩٦ #Cisfun# أَوْدَى

② تَسْتَعْتَرُ مِنْهُ أَوْمَرٌ

③ إِذَا كَتَبَتْ ./bin/z

لَمْ أَنْتَ لَدَمْ تَسْتَدِعِي Execve() لِتَشْتَلِهَا الْبِرَاجِمُ

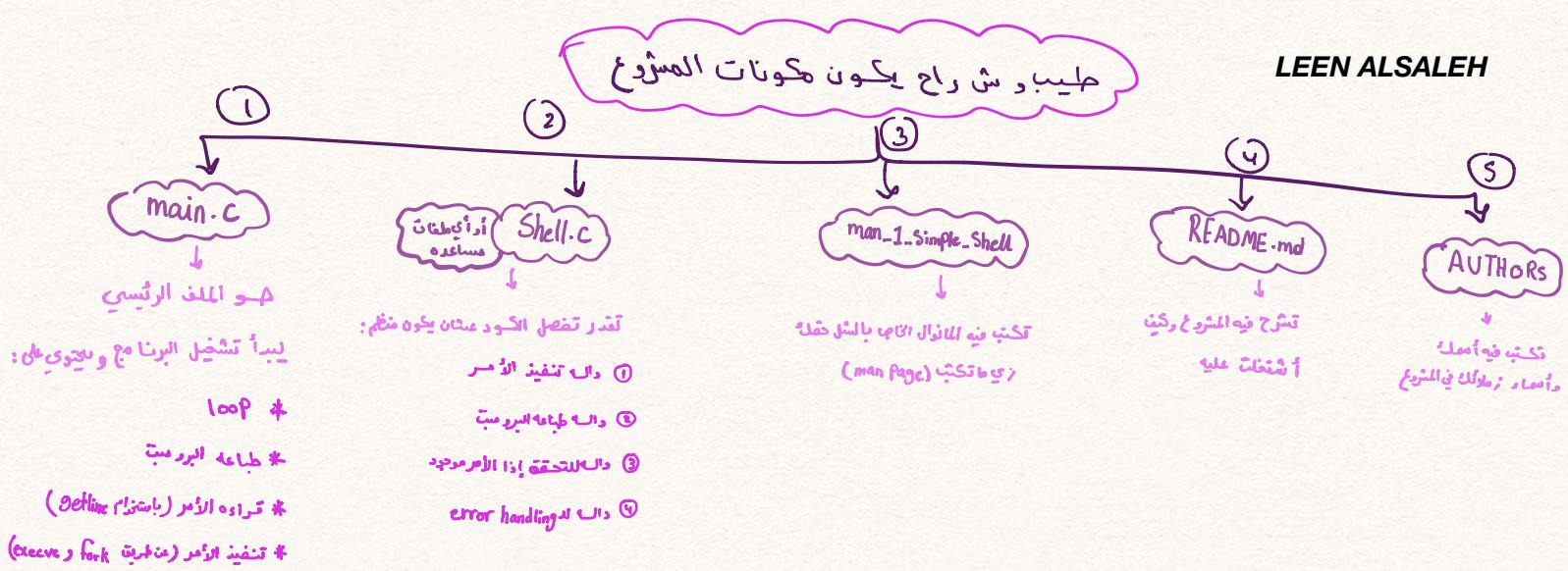
لَمْ لَدَمْ تَسْتَدِعِي fork() عَلَيْهِ تَسْتَدِعِي "عَلَيْهِ جَوِيدَةً" لِتَشْتَلِهِ

لَمْ دَنْتَتْهَارِهَا تَسْتَدِعِي Wait()

④ إِذَا كَتَبَتْ أَوْمَرٌ مِنْهُ مُوجَدٌ مُثُلَّ `xyz`

لَمْ دَرِجَطْعَلَهُ `./hash:xyz: not found`:

⑤ إِذَا خَفَقَتْ Ctrl+D (نَهْيَنِي EOF) - نَهْيَنِي الْأَلْفَدَ : تَطْلُبُ مِنَ الـ Shell بِطَرِيقِ مَسْتَانَهِ



❶ لا نستخدم أكثر من 5 دوال في الملف الرئيسي

❷ تأكد ان الكود نظيف وغير باختصار Betty

❸ أي ملف header لا يحتوي على include guard

❹ اتابع أي شي زيارة عن المتوقع

نهاية مهمه :

2. Simple shell 0.1

mandatory

Score: 100.00% (Checks completed: 100.00%)

Write a UNIX command line interpreter.

• Usage: `simple_shell`

Your Shell should:

- Display a prompt and wait for the user to type a command. A command line always ends with a new line.
- The prompt is displayed again each time a command has been executed.
- The command lines are simple, no semicolons, no pipes, no redirections or any other advanced features.
- The command lines are made only of one word. No arguments will be passed to programs.
- If an executable cannot be found, print an error message and display the prompt again.
- Handle errors.
- You have to handle the "end of file" condition (`Ctrl+D`)

You don't have to:

- use the `PATH`
- implement built-ins
- handle special characters: `^`, `*`, `,`, `\`, `*`, `&`, `#`
- be able to move the cursor
- handle commands with arguments

`execve` will be the core part of your Shell, don't forget to pass the environ to it...

```
julien@ubuntu:~/shell$ ./shell
#isfun$ ls
./shell: No such file or directory
#isfun$ /bin/ls
```

يرجع طبع البروبست مرة ثانية

\$

وكل هذا يعني نفسه كل موه تكتب فيها أمر

الخروج من الشل

أو إذا كنت أنت Ctrl+D أو التهون يفتح
(يعني المستخدم يفتح Enter يعني اللي البرنامج يطبع

أو منتهى تكتب exit بعدد إذا دعوه

ال코드 منطقى ميسه وملائم لخطوات :

```
while (1) {
    Print_Prompt();
    if (fscanf(stdin, "%s", user_input) == EOF || NULL) {
        // User typed Ctrl+D
        // Or EOF or NULL blocks
        // Or read error
        // Read for child to finish
        if (fork() == 0) {
            // Create child
            // Execute command
            // Read for child to finish
            // Waitpid for parent to finish
        }
    }
}
```

تنفذ أوامر بسيطة يكتبها المستخدم مثل /bin/ls ، بدون أي معانين منقحة

(ما في باب او arguments او PATH حتى)

طيب كيف يشنغل الـ Shell

❶ أول شي نشنغل البرنامج

وأنت تفتح المتصفح وتحسب :

/hsh

يستخدم `getline()` حتى يأخذ المسقط الذي يكتبها كامل

❷ ترتيب الأمر (ذا احتاج

* يشيل عادة \n newline الموجوده في نهاية الأمر

* منه يستخدم `Strtok()` أو دالة بيشه عشان تقفل السطر

وتعريف إيه /bin/ls/

❸ تنتهي الأمر:

* يسيوي نسخه هنا نفسه بامتناع

* في الم عليه (Child Process) يستخدم `execve()` لتنفيذ

* أو (Parent) تنتظر حتى تنتهي (Child Process) بامتناع `Waitpid()` أو `Wait()`

❹ النتيجة :

تشوف النتيجه الدعم في المستشه مهل :

file1 file2 main.c

Enter

❺ المستعمل يكتب الأمر :

/bin/ls

الفرق بين النسخة ٠.١ و ٠.٢

3. Simple shell 0.2

Score: 100.00% (Checks completed: 100.00%)

Simple shell 0.1 +

- Handle command lines with arguments

Repo:

GitHub repository: [holbertonschool-simple_shell](#)

لتنفيذ أوامر بسيطة بدون أي arguments (مثل؟ bin/ls)

النسخة

٠.١

تنفيذ أوامر تحتوي على arguments (مثل؟ bin/ls -l /tmp)

٠.٢

كيف يشتغل ٥٩ ٥٦

٧) أنتظر العملية تنتهي

إذا child process ينتظر إما Parent Process
أو ينتظر إما child process يختفي

Wait(NULL);

٨) نشوء النتائج

```
total 8
drwxrwxrwt 2 root root 4096 Apr 16 20:00 tmpfile
```

٩) تكرر العملية

\$ يعني يرجع بطبع :

وليرأ أمر جديداً .

١٥) مثل يدمن أو واعي على الخلفيات انتفخ

١) تحمل هنا : /bin/echo hello world

args[0] = "/bin/echo" : ٤) راج تقسيم :

args[1] = "hello"

args[2] = "world"

args[3] = NULL

{ "n|t| " هي delimiters ملحوظة عند تقطيل محفوظة لا

٣) راج تنفذها باستخان :

execve(args[0], args, environ);

٦) كيف الذر تنفذ مع arguments

* يسو عليه جديده مع fork()

: Child Process في

execve(args[0], args, environ);

* يستخدم args[0] كأمر

* يستخدم arguments كمحفظة لا

* يغير البيئة environ

دائم تأكروا أن آخر عنصر في المعرفة args هو NULL لأن execve() تليه

* ملاحظة

Score: 100.00% (Checks completed: 100.00%)

Simple shell 0.2 +

- Handle the PATH
- fork must not be called if the command doesn't exist

```
julien@ubuntu:~/shell$ ./shell_0.3
:) /bin/ls
barbie_j env-main.c exec.c fork.c pid.c ppid.c prompt prompt.c shell_0.3 stat test_scripting.sh
wait.c
env-environ.c exec fork mypid ppid printenv promptc shell shell.c stat.c wait
:) ls
barbie_j env-main.c exec.c fork.c pid.c ppid.c prompt prompt.c shell_0.3 stat test_scripting.sh
wait.c
env-environ.c exec fork mypid ppid printenv promptc shell shell.c stat.c wait
:) ls -l /tmp
total 20
-rw-r--r-- 1 julien julien 0 Dec 5 12:09 config-err-aAMZr
drwx----- 3 root root 4096 Dec 5 12:09 systemd-private-062a0eca7f2a44349733e78cb4abfff4-colord.service-V7DUzr
drwx----- 3 root root 4096 Dec 5 12:09 systemd-private-062a0eca7f2a44349733e78cb4abfff4-rtkit-daemon.service-A
NGvO
drwx----- 3 root root 4096 Dec 5 12:07 systemd-private-062a0eca7f2a44349733e78cb4abfff4-timesyncd.serv
ice-CXUH
-rw-rw-r-- 1 julien julien 0 Dec 5 12:09 unity_support_test.0
:) ^C
julien@ubuntu:~/shell$
```

الفرق بين النسخة 0.3 و 0.2

النسخة	نقطة نظر
0.2	يُدعم تفسيت أوامر مع arguments
0.3	* يُدعم أوامر من الـ PATH
	* عدم استخدام fork() إذا الأمر غير مجد

سؤال !! أيّن هو بكل موضع PATH ؟

PATH هو متغير يحتوي على مسارات للمحميات التي فيها أوامر نظام (مثل cat, echo, rm, داد إلخ...)

PATH = /user/local/bin:/user/bin:/bin

طبع ومن يعني هنا الكلام هذ

* إذا كانت دا ، النظام يدور عليه بـ الترتيب :

- user/local/bin/ls/ .
- user/bin/ls/ .
- bin/ls/ .

كيف المستو الذي يكون الكلام هنا هذ

① نشخل البرنامج

./shell_0.3

: Prompt <-- مدخل

(:)

③ المستخدم يكتب أمر من غير مسار كامل :

ls -l /tmp

④ Shell يدخل السطر إلى كلمات ذي

```
args[0] = "ls"
args[1] = "."
args[2] = "/tmp";
args[3] = NULL;
```

⑤ كيت تستأكِر

- * إذا كان نفس الأمر فيه مسار كامل مثل /bin/ls .
- * وإن يتم تفسيته بـ باعتره
- * إذا كان أمر ماوري بدون مسار مثل --ls :
- * البرنامج يدور عليه في مسارات PATH .
- * وهو كل كيت يصير هذ

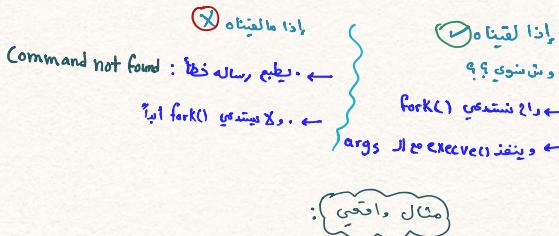
٥) يتم هذا عن طريق :
getenv ("PATH") .

• تقسيم PATH إلى أجزاء بـ strtok

• توكي كل جزء مع الأسم الأوامر :

"/bin/" + "/ls" ⇒ "/bin/ls"

نتأكد من وجود العنوان باستخدام access (full_path, X_OK)



٦) هل وافق :

المستخدم يكتب : echo Hello shell

PATH=/usr/local/bin:/usr/bin/bin : يبحث Shell

... يبحث

⑧ user/local/bin/echo/ .
 ↳ موجود (✓) user/bin/echo/ .

طبع هذ

وإن شئت تفسيت fork() :

Hello shell :

:

ولوحة يطبع :

لذا كان أمر عزيز مويد هذ

notCommand : Command not found .

انتبه : ما أمر fork() لا يتيه بنطع خطأ

* ملاحظة : في هذه النسخة ، Shell حار ذاتي :

* يشتق الأوامر من

* يختبر هوارد الجماز (ما ي Sociology بـ داد)

* أكذب نتبه : bash حقيقى

5. Simple shell 0.4

Score: 100.00% (Checks completed: 100.00%)

Simple shell 0.3 +

- Implement the `exit` built-in, that exits the shell
- Usage: `exit`
- You don't have to handle any argument to the built-in `exit`

Repo:

- GitHub repository: [holbertonschool-simple_shell](#)

و س الفرق بين ٠.٣ و ٠.٤

النسخة	
٠.٣	ينفذ أوامر من <code>PATH</code> و يتحقق من وجود الأمر قبل <code>(fork()</code>
٠.٤	يدعم الأمر المخرج <code>exit</code>
٠.٤	المطلوب في المطلوب في

و س المطلوب ٥٩

٥٩ المطلوب في

• تنفيذ أمر جديء اسمه `exit` (هو أمر داخلي - `built-in`)

• لما المستخدم يكتب `exit` يخرج إلى Shell وينتهي البرنامج

• ماتحتاج تتعامل مع `exit1` ، `exit2` فقط `exit` بده أرقام

كيف الكس عم هذا ينتفذ ٥٩

① المستخدم يشغل الـ `Shell`
`/shell_0.4` :

② المستخدم يكتب أمر عادي :
`ls`

نفس اللي قلناه في ٥.٣

• يبحث عن `ls` في `PATH`
• وينفذ الأمر
• ويخرج بطبع : `ls`

③ المستخدم يكتب :
`exit`

وسراح بصير في الكود ٥٩

① البرنامج يقرأ الأمر كالعادة (`getline`)

② يحلله إلى كلمات (`tokenize`)

③ ينشئ على أول كلمة :

```
if(strcmp(args[0], "exit") == 0)
{
    exit(0);
}
```

أنهى البرنامج → ٣

النحو احتبار لسرع

٥٩ النتيجة

• البرنامج يطلع فوراً بدون طاقة زراعة

• مايسوي `execve()` أو `fork()`

• فترب خرج من الـ `loop` الأدمسية

ومن المفترض أن يكون

6. Simple shell 1.0

Score: 100.00% (Checks completed: 100.00%)

mandatory

Simple shell 0.4 +

- Implement the `env` built-in, that prints the current environment

```
julien@ubuntu:~/shell$ ./simple_shell
$ env
USER=julien
LANGUAGE=en_US
SESSION=ubuntu
COMPIZ_CONFIG_PROFILE=ubuntu
SHLVL=1
HOME=/home/julien
_C_IS_Fun.:)
DESKTOP_SESSION=ubuntu
LOGNAME=julien
TERM=xterm-256color
PATH=/home/julien/bin:/home/julien/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
DISPLAY=:0
$ exit
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: [holbertonschool-simple_shell](#)

النسخة

تنفيذ أوامر `exit` ودعم الأمر `PATH`

0.4

تنفيذ `env` جيد

1.0

إرشاد المطلوب

① تنفيذ أمر `env` (built-in - Command) جيد اسمه :

② لما المستخدم يكتب `env` في Shell يعرض كل المتغيرات البيئية (Environment Variables) الجديدة حاليًّا

③ بعدها يرجع له `Prompt` كالعتاد

ملاحظة : ماتحتاج تستخدم `execve` أو `fork` هنا لأنه أمر داخلي

الميزات

النسخة

ينفذ أوامر بسيطة	0.1
يُعمَل على argument	0.2
العامل مع PATH	0.3
built-in : exit	0.4
built-in : env	1.0

كيف تطبق الكلمات هنا؟

① المستخدم يدخل `/simple-shell` : Shell

② المستخدم يكتب أمر `env` :

إيش يصير في الكور؟

① إن Shell يقرأ السطر كواحدة (getline)

② يعلمه إلى ما (args[0] = "env") : tokens

③ يتحقق هل الأمر `env`

```
if (strcmp(args[0], "env") == 0)
{
    Print_env(); → Environment Variables
    Continue;
}
```

كيف تطبع

النسخة

كل المتغيرات البيئية
تقطيع سطر بسطر
يبيدها يرجع إلى Prompt
عندما المستخدم يكتب
أمر جديد

execve أو fork ماتحتاج `env`.

هو أمر داخلي يعنيه Shell.

الآن ما رعنك Shell 2 ينتمي مع `env` و `exit`.

```
$ env
USER=julien
LANGUAGE=en_US
SESSION=ubuntu
...
PATH=/user/bin:/bin:/user/local/bin
DISPLAY=
```

```
extern char **environ;
Void Print_env (void)
{
    int i = 0;
    while (environ[i])
    {
        printf ("%s\n", environ[i]);
        i++;
    }
}
```