ECS 150 - Project #1 - Part 2

Prof. Joël Porquet-Lupine

UC Davis - SQ25

Simple shell

Goal

- Understand important UNIX system calls
- Continue implementing a simple shell called **sshell**

Specifications Part 1 -- Solo (done)

- Execute commands with arguments

```
sshell@ucd$ date -u
```

Specifications Part 2 -- Group (now!)

- Redirect standard output of command to file

```
sshell@ucd$ date -u > file
```

- Pipe the output of commands to other commands

```
sshell@ucd$ cat /etc/passwd | grep root
```

- Offer a selection of builtin commands

```
sshell@ucd$ cd directory
sshell@ucd$ pwd
/home/jporquet/directory
```

- Two extra features (input redirection + background job)

Simple shell

Standard output redirection: `>`

```
sshell@ucd$ echo Hello world>file
+ completed 'echo Hello world>file' [0]
sshell@ucd$ cat file
Hello world
+ completed 'cat file' [0]
```

- Output redirection means that the process's output will be written to a file instead of to the terminal
- Spacing shouldn't matter
  - `echo Hello world>file` is equivalent to `echo Hello world > file`

Simple shell

Pipeline of commands: |

```
sshell@ucd$ echo Hello world | grep Hello|wc -l
1
+ completed 'echo Hello world | grep Hello|wc -l' [0][0][0]
```

- Interconnection of multiple commands into a *job*
- Output of command before '|' is redirected as the input of the command located right after
- Up to three pipes on the same command line

Simple shell

Extra feature #1

Standard input redirection

```
sshell@ucd$ cat file
titi
toto
+ completed 'cat file' [0]
sshell@ucd$ grep toto<file
toto
+ completed 'grep toto<file' [0]
```

- Input redirection means that the process will read the input from a file instead of the keyboard

Simple shell

Extra feature #2

Background job

```
sshell@ucd$ sleep 1&
sshell@ucd$ sleep 5
+ completed 'sleep 1&' [0]
+ completed 'sleep 5' [0]
```

- Ampersand sign indicates that the command should be run in the background
  - Only one background job required
- Shell should not wait for completion before printing a new prompt and accepting a new command
- After the command completes, shell displays the information message when the next prompt is displayed

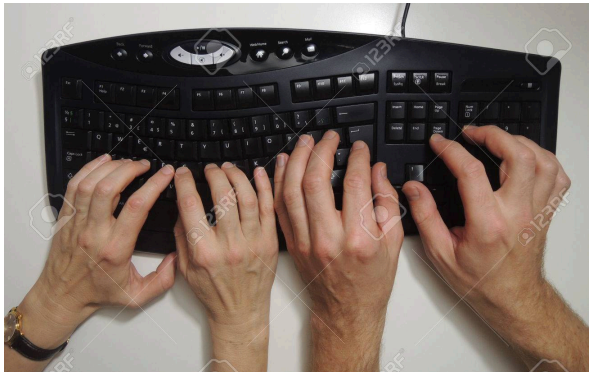
General information

Logistics

- Published this morning, due in 2 weeks, by Thursday, April 24th
- Part 2 is a group effort

Group work

- Teams of exactly two partners
  - Find a partner after/before class or using "Course Chat" on CourseAssist
- Find a partner with whom you can work well
  - Define what kind of collaboration you're looking for before pairing up
  - How to meet? How regularly? Etc.
- Look into and use effective group programming approaches
  - E.g., pair programming



Git

Introduction

> *Version control system for tracking changes in computer files and coordinating work on those files among multiple people.*

Unlimited private repositories on github and bitbucket.

Initial configuration

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email "name@ucdavis.edu"
```

Git

How to start?

1. Create account and **private** repository online
2. Add partner as collaborator
3. Clone it locally

```
$ git clone git@github.com:nickname/ecs150-sshell.git && cd ecs150-sshell
```

4. Start coding

```
$ vim sshell.c
```

5. Commit and push

```
$ git add sshell.c
$ git commit -m "Initial commit"
$ git push
```

6. Your partner can now pull your commit

```
$ git pull
```

More resources

- https://guides.github.com/activities/hello-world/
- https://www.atlassian.com/git/tutorials

Best practices

Submit production code

```
int main( void)     {
  func1(a, b  , c);
       func2(a,b,c);
    // printf("Back from func2\n");
    // Still need to investigate func3()
     func3();
    /* Also need to make a haircut appt */

            // Return value 0
    return 0;
 }
```

```
myexec: mycode.c
    gcc -g -Wall -Werror ...
```

Not production ready :(

```
int main(void)
{
    /* Parse the command line */
    func1(a, b, c);
    func2(a, b, c);

    /* Run requested command(s) */
    func3();

    return 0;
}
```

```
myexec: mycode.c
    gcc -O2 -Wall -Werror ...
```

Production ready!

- What happens in development "stays" in development
- Production code (i.e., your submission) is a final product, not a draft
- Remove *dead code* and debugging comments
- Compile for performance, not debug

Coding tips

Fixed values (1)

```c
int main(void)
{
    char cmdline[512];

    fgets(cmdline, 512, stdin);
    ...

    return 0;
}
```

```c
#define CMDLINE_MAX 512

int main(void)
{
    char cmdline[CMDLINE_MAX];

    fgets(cmdline, CMDLINE_MAX, stdin);
    ...

    return 0;
}
```

- Avoid hardcoded values and prefer *named macros*
- Definition in one location, easy to update
- Meaning of value is more important than value itself

Coding tips

Fixed values (2)

```
void error_message(int error_code)
{
    switch(error_code) {
        case 0:
            fprintf(stderr,
                    "Error: missing command\n");
            break;
        case 1:
            fprintf(stderr,
                    "Error: command not found\n");
            break;
        case 2:
            ...
    }
}

void func(void)
{
    ...
    error_message(1);
    ...
}
```

- Again, avoid hardcoded values
- Use generic constructs

```
enum {
    ERR_MISSING_CMD,
    ERR_CMD_NOTFOUND,
    ...
};

void error_message(int error_code)
{
    switch(error_code) {
        case ERR_MISSING_CMD:
            fprintf(stderr,
                    "Error: missing command\n");
            break;
        case ERR_CMD_NOTFOUND:
            fprintf(stderr,
                    "Error: command not found\n");
            break;
        ...
    }
}

void func(void)
{
    ...
    error_message(ERR_CMD_NOTFOUND);
    ...
}
```

Coding tips

Error checking

```
int myfunc(char *buffer, int len)
{
    int i;

    if (buffer) {
        if (len >= 0) {

            /* Process buffer */
            for (i = 0; i < len; i++)
                ...

            return 0;
        }
    }

    return -1;
}
```

```
int myfunc(char *buffer, int len)
{
    int i;

    /* Error checking */
    if (!buffer || len < 0)
        return -1;

    /* Process buffer */
    for (i = 0; i < len; i++)
        ...

    return 0;
}
```

- Avoid deep nesting of code
- Process error cases first, then proceed with actual function's functionality, at the same indentation level

Coding tips

Function order

```c
#include <stdio.h>

int func1(void);
char func2(int);
int func3(char);

int main(void)
{
    func3();
}

int func1(void)
{
    func2();
    func3();
}

char func2(int)
{
    func3();
}

int func3(char)
{
}
```

```c
#include <stdio.h>

int func3(char)
{
}

char func2(int)
{
    func3();
}

int func1(void)
{
    func2();
    func3();
}

int main(void)
{
    func3();
}
```

- Popular design pattern in big C project, such as the Linux kernel
- Less code, less prototypes to maintain
- With a proper code editor function navigation is easy
    - Look into `ctags` or `cscope` or `LSP` for your code editor

Coding tips

Functions

```c
#include <stdio.h>

int main(void)
{
    /* 1. First we do this */
    ...
    /* 2. Then we do that */
    ...
    /* 3. Then it depends */
    if (value == 42) {
        /* 3a. Some code using 42 */
    } else {
        /* 3b. Same code not using 42 */
    }

    return 0;
}
```

```c
#include <stdio.h>

int func1(char)
{
}

char func2(int)
{
}

int func3(int param)
{
}

int main(void)
{
    func1();
    func2();
    func3(value);
}
```

- Split your code into tinier chunks
  - If a function starts doing multiple things, time to break it down
- When possible, write generic and parameterizable functions

Coding style tips

Editor configuration

What you might see in your code editor:

```
struct linked_list {

  struct linked_list_node *head;
  int size;

};
```

What I see in mine:

```
struct linked_list {

  struct linked_list_node *head;
        int size;

};
```

- Ensure visual consistency between code editors
  - Spaces vs tabs
  - Settle for one convention with your partner, and configure your editors accordingly
- Remove unnecessary spaces
  - Take actual space in the source code

Good luck!