Matt Forbes
CS352
Assignment 2
October 5, 2011

```
Script started on Wed 05 Oct 2011 12:12:46 PM PDT
[4mCF405-10[24m:[1m~/school/352/as2[0m% ls

arg_parse.c  builtin.c  msh  msh.c  pdf  proto.h  run.txt
[4mCF405-10[24m:[1m~/school/352/as2[0m% rm ./msh

[4mCF405-10[24m:[1m~/school/352/as2[0m% gcc -Wall -o msh *.c

[4mCF405-10[24m:[1m~/school/352/as2[0m% ./msh

% "ls"
arg_parse.c  builtin.c  msh  msh.c  pdf  proto.h  run.txt
%   "ls" -"a""l"
total 27
drwxr-xr-x 3 forbesm2 students     9 2011-10-05 12:12 .
drwxr-xr-x 4 forbesm2 students     5 2011-10-04 14:04 ..
-rw-r--r-- 1 forbesm2 students  1885 2011-10-04 17:13 arg_parse.c
-rw-r--r-- 1 forbesm2 students  1316 2011-10-04 17:09 builtin.c
-rwxr-xr-x 1 forbesm2 students 13697 2011-10-05 12:12 msh
-rw-r--r-- 1 forbesm2 students  1619 2011-10-04 17:04 msh.c
drwxr-xr-x 2 forbesm2 students     7 2011-10-05 12:10 pdf
-rw-r--r-- 1 forbesm2 students   135 2011-10-04 15:36 proto.h
-rw-r--r-- 1 forbesm2 students     0 2011-10-05 12:12 run.txt
% mo"re" pro     "pro"to.h
/* argugment parsing */
int arg_parse(char *line, char ***argvp);

/* builtin commands */
int try_builtin(int argc, char **argv);

% aecho now test aecho
now test aecho
% aecho "arg'  1" and "arg3 is here"
arg1 and arg3 is here
% aecho "  -n "no new line followed by emy  pty input (blank line)"
no new line followed by empty input (blank line)%
%
% aecho "previos  us tow    wo inputs were blank line"  s  s"
previous two inputs were blank lines
% test e                    aecho "now test exit"
now test exit
% exit 2
[4mCF405-10[24m:[1m~/school/352/as2[0m% echo $status

2
[4mCF405-10[24m:[1m~/school/352/as2[0m% ^D   exit

Script done on Wed 05 Oct 2011 12:16:04 PM PDT
```

```c
/* argugment parsing */
int arg_parse(char *line, char ***argvp);

/* builtin commands */
int try_builtin(int argc, char **argv);
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include "proto.h"

struct builtin {
    char *name;
    void (*fn)(int, char **argv);
};

void bin_exit(int argc, char **argv) {
    int code = (argc == 1) ? 0 : atoi(argv[1]);
    exit(code);
}

void bin_aecho(int argc, char **argv) {
    int print_nl = 1,
        i = 0;

    if (argc > 1 && strncmp("-n", argv[1], 2) == 0) {
        print_nl = 0;
        /* don't print the -n */
        argv++;
        argc--;
    }

    for(i = 1; i < argc; i++) {
        if (i != 1) {
            if (write(1, " ", 1) < 0)
                perror("write");
        }

        if (write(1, argv[i], strlen(argv[i])) < 0)
            perror("write");

    }

    if (print_nl) {
        if (write(1, "\n", 1) < 0)
            perror("write");
    }

}


const int NUM_BUILTINS = 2;
const struct builtin BUILTINS[] = {
    { "aecho", bin_aecho },
    { "exit", bin_exit }
};

int try_builtin(int argc, char **argv) {
    int i;
    char *name = argv[0];

    for (i = 0; i<NUM_BUILTINS; i++) {
        if (strcmp(name, BUILTINS[i].name) == 0) {
            /* matched builtin name, call it */
            BUILTINS[i].fn(argc, argv);
            return 1;
        }
    }
    /* didn't match */
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

#include "proto.h"


/* parses a line read from shell into individual args. */
int arg_parse(char *line, char ***argvp)
{
    char   *c, *ptr;
    char   **it;
    int    argc = 0, reading = 0;

    /* determine number of args */
    ptr = line;
    while ( *ptr ) {
        /* eat spaces */
        while (*ptr && *ptr == ' ')
            ptr++;

        /* handle possible trailing spaces */
        if (!*ptr)
            break;

        /* arg starts here */
        argc++;

        while ( *ptr && *ptr != ' ') {
            /* now have either quote or other charater */
            if (*ptr == '"') {
                ptr++;
                /* read to end quote */
                while (*ptr && *ptr != '"')
                    ptr++;
                ptr++;
            } else {
                ptr++;
            }
        }

    }

    /* create array for args */
    *argvp = (char **)malloc(argc * sizeof(char *) + 1);
    if (!*argvp)
        perror("malloc");

    /* partition args and store positions of first characters */
    it = *argvp;
    reading = 0;
    c = ptr = line;
    while ( *ptr ) {
        /* eat spaces */
        while (*ptr && *ptr == ' ')
            ptr++;

        /* arg starts here */
        *(it++) = c;

        while ( *ptr && *ptr != ' ') {
            /* now have either quote or other charater */
            if (*ptr == '"') {
                ptr++;
                /* read to end quote */
                while (*ptr && *ptr != '"')
                    *(c++) = *(ptr++);
                ptr++;
```

```
            } else {
                *(c++) = *(ptr++);
            }
        }

        /* null terminate argument */
        ptr++;
        *(c++) = 0;

    }

    /* add null pointer to end of array */
    *it = (char *)0;

    return argc;
}
```

```c
/* CS 352 -- Mini Shell!
 *
 *   Matt Forbes - Assignment 1 - 9/23/11
 *
 */

#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

#include "proto.h"

/* Constants */

#define LINELEN 1024

/* Prototypes */

void processline (char *line);

/* Shell main */

int main (void)
{
    char   buffer [LINELEN];
    int    len;

    while (1) {

        /* prompt and get line */
        fprintf (stderr, "%% ");
        if (fgets (buffer, LINELEN, stdin) != buffer)
            break;

        /* Get rid of \n at end of buffer. */
        len = strlen(buffer);
        if (buffer[len-1] == '\n')
            buffer[len-1] = 0;

        /* Run it ... */
        processline (buffer);

    }

    if (!feof(stdin))
        perror ("read");

    return 0;           /* Also known as exit (0); */
}


void processline (char *line)
{
    pid_t  cpid;
    int    status,
           argc;
    char   **argv;

    argc = arg_parse(line, &argv);

    /* when no arguments, do nothing */
    if (argc == 0)
        return;
```

```
    /* try calling a builtin, return if successful */
    if (try_builtin(argc, argv))
        return;

    /* Start a new process to do the job. */
    cpid = fork();
    if (cpid < 0) {
        perror ("fork");
        return;
    }

    /* Check for who we are! */
    if (cpid == 0) {
        /* We are the child! */
        execvp(argv[0], argv);
        perror ("exec");
        exit (127);
    }

    /* free argv when parent */
    free(argv);

    /* Have the parent wait for child to complete */
    if (wait (&status) < 0)
        perror ("wait");
}
```