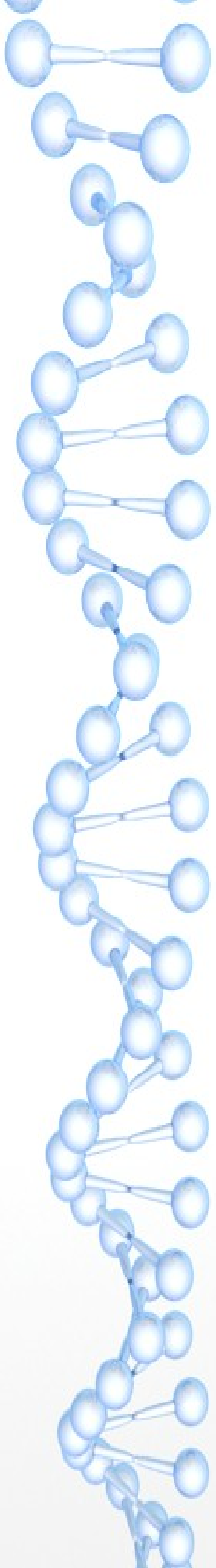
UNIX Command Line Arguments

Northern Illinois University
CSCI 330
Jon Lehuta



Introduction

- When you type a command at the shell, the shell breaks it up into strings separated by spaces.
  - `command arg1 arg2 arg3 ...`
- Quotes can allow you to specify strings that contain spaces, backslashes can also be used to escape the spaces.
  - `command “arg1 with spaces” arg2 arg3 ...`
  - `command 'arg1 with spaces' arg2 arg3 ...`
  - `command arg1\ with\ spaces arg2 arg3 ...`

Handling within a program

- The shell will provide the executed program with a list of all of the command line parameters. In C/C++, the relevant information is passed into the program's **`main`** function as *argc* and *argv*.
- `int main(int argc, char *argv[])`
- *argc* – number of arguments (short for argument count)
- *argv* – this is an array of C strings, one string per command line argument.

Within C/C++ Program

- Argc will be one greater than the actual number of arguments, because the name of the program as run is passed as argv[0].
- So, if you were to take the command from before:
  - `command arg1 arg2 arg3 ...`
- You would see the following values:
  - `argc=4;`
  - `argv[0] = “command”;`
  - `argv[1] = “arg1”;`
  - `argv[2] = “arg2”;`
  - `argv[3] = “arg3”;`

C/C++ Continued

- If you'd like to test it out yourself, use this code to print out all of the command line arguments in the same format as the last slide.

```
int main(int argc, char *argv[]) {
  for(int i=0;i<argc; ++i) {
    cout “argv[“ << i << “]=\””;
    cout << argv[i] << “\”” << endl;
  }
}
```

Are we there yet?

- These are easy to deal with as long as you can ensure that the user puts things in the right position.
- One place that this may not happen is for programs that have various optional parameters.

The Problem with Options

- In one of your assignments, the program is required to have an optional parameter, "-c". There are also several other parameters.
- Let's take a look at the ways this program could be run, assuming three positional parameters are specified:
  - `command -c arg1 arg2 arg3`
  - `command arg1 -c arg2 arg3`
  - `command arg1 arg2 -c arg3`
  - `command arg1 arg2 arg3 -c`
  - `command arg1 arg2 arg3`
- The shell will still break this up based on spaces, so for the first version, you'd have *argv*[1]="-c", *argv*[2]="arg1", *argv*[3]="arg2", *argv*[4]="arg3"
- Notice that the order is different than it would have been without the "-c" parameter, where we end up with *argv*[1]="arg1", *argv*[2]="arg2", *argv*[3]="arg3".
- This will obviously cause problems if you expect them in order.

The Solution (**`getopt`**(3))

- Fortunately, this problem was solved a long time ago – use the getopt function.
- `int` **`getopt`**`(int` *`argc`*`, char * const` *`argv`*`[], const char *`*`optstring`*`);`
- *`argc`* – pass along the argc from your main
- *`argv`* – pass along the argv from your main
- *`optstring`* – a formatted string listing all of the allowed options.
- Every time you call this function, it will return the ASCII value of the first optional parameter found (so you can act on it), moving that parameter to the beginning of `argv`, shuffling the non-option pointers forward, and incrementing the global variable `optind`. When it no longer finds any, it returns -1, and `optind` will be the position of the first non-option parameter.

An Example

```
// Sample code to handle the -c optional parameter
int main(int argc, char *argv[]) {
  int opt; bool cflagused=false; char optstring[] = “c”;
  // Each step of this loop handles one optional parameter
  while((opt = getopt(argc, argv, optstring)) != -1) { // end when -1 returned
    switch(opt) { // make a case for each option supported here
      case 'c':
        // run any option-specific code now
        // here I set a boolean flag to check later on
        cflagused=true; break;
    }
  }
  // When the code reaches here, the -c option
  // will no longer be in the argc, argv list
  ...
}
```

getopt_long

- **getopt** works for single-character options. If you want to work with longer ones, the **getopt_long** function can help. Check the **getopt** manpage for details.