Arboreal User Guide

Table of Contents

Installing Arboreal

Arboreal is currently not integrated with the kernel and as such runs similarly to a virtual file system albeit with a more experimental structure. Future work will be focused on direct integration with the kernal in order to provide more traditional usability. In the meantime playing around with and testing the file system can be achived through a few easy steps:

1. **Download the project**
2. **Changed directory (within the shell) to the folder within the project hierarchy named `Source`**
3. **Type `make` into your shell**
4. **You will now need to first run the daemon process. This process intercepts al communication attempts with the File System and will execute functions accordingly. There are a number of command line arguments that can be passed to the daemon:**
    - `-d` **This flag is used to tell the daemon to enable debugging**
    - `-v` **This flag is used to tell the daemon to return file information (such as that returned by a call to find) with as much information as possible. Omitting this flag will cause the daemon to return a reduced version of file information**
    - **You may enable either of these options or both (input order does not matter, that is `-d -v` will work the same as `-v -d`)**
5. **Finally, Simply Type `./daemon` followed by your chosen flag or flags (be sure to include a space in between). For example, if I wanted to run the daemon with verbose file information and debugging enabled the command would look like: `./daemon -v -d`**

At this point you'll be ready to move on to the next step, starting the command line or GUI interface. Notice that the daemon does not output anything to the screen as it is running. **This is OK!** Its whole purpose is to be a background process that aids communication with the file system. **If you decided to enable debugging, the output will be located in a file called `Arboreal.log` .**

A final note:
By typing make, the "disk" will be formatted for you with the default values and partition names/ counts. It is possible to change these to better suit your needs however it is a little bit more involved.

1. **Open and edit a file called `diskInfo.d` (It is located in the `Source` folder)**
2. **Using the following syntax, edit the file as you see fit:**
    - **Line 1 needs to always be:**
      `Diskfile name, number of blocks on disk, size of each block in bytes, number of partitions` (Omit the commas in favor of spaces)
    - **Lines 2 - X need to always be:** `Partition name, number of blocks in the partition, maximum filename size` (Again omit commas in favor of spaces)
    - **There are restrictions on values for `diskInfo` :**
        - **Filename sizes** are restricted to **no more than** `block size / 2`
        - **Block size must be a power of 2**
        - There exists a **hardcap on the number of tags that can be associated with a file** and it is equal to `( ( ( block size – filename size – 136 ) / sizeof( BlkNumType ) ) + ( block size / sizeof( BlkNumType ) )`
        - The **maximum possible blocksize** is `16 Kb`
3. **Next you will have to open `daemon.cpp` (it is located under `Source/Filesystem/`) and edit this line of code:**
    - `d = new Disk(#1, #,2 const_cast<char *>("diskfile_name"));`
    - **Change `#1` to whatever value you picked for `number of blocks` in `diskInfo.d` .**

        - So if I decided that I wanted 4000 blocks I would type `4000` in for `#1` (**The number here and in** `diskInfo.d` **MUST MATCH**).
    - **Change** `#2` **to whatever value you picked for** `block size in bytes` **in** `diskInfo.d`
        - So if I decided that I wanted blocks to be 4096 bytes large I would type `4096` in for `#2` (**Once again I stress that the number here and in** `diskInfo.d` **MUST MATCH**)
    - **Finally, change** `diskfile_name` **to the name of the disk file you chose in** `diskInfo.d`
4. **You are now almost ready, the final step is to type** `make clean` **followed by** `make` **in the shell and run the through the same steps as above for starting the daemon**
5. **You are now good to go!**

Starting The Command Line

Before beginning anything below, make sure that a daemon process (and ONLY ONE daemon process) is running, if the command line cannot connect to the daemon process it will quit on startup with an appropriate error message

The command line utility has multiple optional arguments but it does contain a single mandatory argument. This is the **Partition Name** that the command line will be working on. If no partition name is given the command line will fail on startup with an appropriate error message. Additionally it is important to note that **the partition that is given to the command line must already exist**. If it does not, an appropriate error will be thrown. Finally, **it does not matter if the partition is already in use by another command line and it does not matter how many command lines are currently active**, in both cases you will still be able to work wth the file system (provided that the partition you gave exists). After providing the partitoon name you are free to run the command line. However, should you wish to, there are some optional command line arguments:

- `-d` **This argument will enable debugging for both the command line and the liaison process**
- `-s` **This argument will alert the command line that input will be coming from a file rather than a user**
- `-s -d` **This will enable debugging for the command line AND alert it to the fact that input will be coming from a file rather than a user**

For example, if i wished to pipe input from a file to the command line and enable debugging, I would run the command line process like so:
`./commandline PartitionName -s -d < some_random_file.ext`

(*Note that* `./commandline -d -s < some_random_file.ext` *will not work, that is, make sure the debug flag comes last!*)

But if I wanted to just enable debugging and read from user input, I would run the command line process like so:
`./commandline PartitionName -d`

At this point you should see the arboreal header and `Arboreal >>` indicating that the command line is ready to accept input. **To send input to the command line simply type the command you wish to execute** (see *Valid Commands And Their Syntax* section for commands or type `help` or `h`) **and press enter.**

Note
If you chose to enable debugging for the command line, all debug output will be written to a file named `Arboreal.log`. Do not worry if this file does not yet exist, it will be created for you on startup.

Valid Commands And Their Syntax

Before we begin please make note of a few things:

1. **There is a maximum command size of `4096 bytes`** Although a lot of these commands have support for defining multiple operations within a single command (such as search for multiple files by name without having to enter a seperate `find -f` command for each file), if the total size of the command exceeds the current default of `4096 bytes`, the command will fail to send and be ignored.
2. **DO NOT include spaces between `list` and/or `set` items or between items and `{}` or `[]`.**
3. Commands of the form `../tag/tag/file` or `../tag/tag` only **emulate** directory structures. In fact, **Arboreal does not have the concept of a directory.** However, this style of syntax is universally understood and easy to type and so was maintained.

"Help" Commands

```
Arboreal >> help
Arboreal >> h
```

These two commands will bring up a helper subprocess which will display a list of the command archetypes and show the user the specific commands (and their syntax) that are housed under each archetype. The helper subprocess continues running until the user decides to quit it.

```
Arboreal >> -h --command_archetype

e.g.
Arboreal >> -h --find
```

This version of the help command will show the usage for a single command archetype. (Unlike the `help` or `h` commands it will not start a "helper" subprocess but will simply display the usage for the particular archetype and await the next file system command)

"Quit" Commands

```
Arboreal >> quit
Arboreal >> q
Arboreal >> Q
```

All of these will attempt to terminate the current command line process. This command does not affect other concurrently running command lines it will only quit the currently active command line process. The user must confirm the quit before the command will actually be executed. this is to prevent accidental quits. The quit commands are built with proper cleanup in mind and should not leave any junk behind.

Changing The Current Working Directory

```
Arboreal >> cd tagnameX/.../tagnameXn
Arboreal >> cd ./tagname1/tagname2
```

Changing "directories" is quite easy and supports relative paths and absolute paths.

- Version 1 will change to the "directory" which includes files tagged with all of the tags included in the command. That is, if you were to `cd Documents/December/Papers` all files "within" the "directory" would be tagged with `Documents, December, Papers`.
- Version 2 will append whatever you typed to the current working directory. **You MUST include the period**. For example if I'm currently in the "directory" `Documents/December/Papers` and I `cd ./CreativeWriting/BeerBrawls`, my resultant "directory" will be `Documents/December/Papers/CreativeWriting/BeerBrawls`.

"Find" Commands

Find Files By Tag

```
Arboreal >> find -t [tagnameX,...,tagnameXn]
Arboreal >> find -t {tagnameX,...,tagnameXn}
Arboreal >> find -t [tagnameZ,{tagnameX,...,tagnameXn},...,tagnameZn]
Arboreal >> find -t {tagnameZ,tagnamZ1,[tagnameX,...,tagnameXn],...,tagnameZn}
```

This command searches for files by tag. It is quite powerful and allows you to search for any combination of tags. Commands that use `{}` are called `sets` and will tell the file system to **search for ALL files which are tagged with ALL of the specified tags.** You can think of this as a bunch of `&&` operations, that is, you want a file tagged with :

`{ this tag, and this tag, and this tag, ... etc}`

Commands that use `[]` are called `lists` and will tell the system to **search for ANY file which is tagged with ANY of the tags specified.** You can think of this as a bunch of `||` operations, that is, you want a file tagged with:

`[this tag, or this tag, or this tag, ... etc]`

What's great is that you can actually nest any of these within one another! Although nesting a bunch of `sets` or `lists` won't be any diffferent from simply using one big list or set (i.e. `[t1,[t2,t3,t4]]` is the exact same as `[t1,t2,t3,t4]` this goes for `sets` as well). However, tings get interesting when you pass a command such as:

`find -t [tag1,tag2,{tag45,tag78,[tag9,tag10],tag5},tag100]`

This particular command will search for any file with:

```
    ( tag1 )
    -or- ( tag2 )
    -or- ( tag100 )
    -or- ( tag45 & tag78 & tag9 & tag5 )
    -or- ( tag45 & tag78 & tag10 & tag5 )
```

(*Of course you accomplish similar things even with a command that is a `list` nested within a `set` rather than this example which is a `set` nested within a `list`*)

As you can see, nesting these operations creates some really powerful search options!

Find Files By Name

```
Arboreal >> find -f [filenameX(.ext),...,filenameXn(.ext)]
```

This command will (as the title suggests) find a file based on its name. In order to make search for multiple files easier, you have the option of defining several filenames to search for. For example:

```
Arboreal >> find -f [BookReport.pdf,ExperimentData.numbers,Makefile]
```

Will search for a file named `BookReport.pdf`, a file named `ExperimentData.numbers`, and a file named `Makefile`. **This command does not support nested `[]` or `{}` as these have no meaning when searching for files by name.**

It is also important to note that, while file extensions are optional for this command, a search for `BookReport` will NOT return `BookReport.pdf` and a search for `BookReport.pdf` will NOT return `BookReport`.

Both finding files by tag and by name will return file information and the toal number of files found. If the file could not be found or the tag exists but is completely empty, a message stating so will be returned instead. If the results are a mixure of non existant and existant files, the appropriate message will be printed for each file (either the file information or "not found" message) and a count of the total number fo files found will still be displayed.

"New" Commands

Creating New Tags

```
Arboreal >> new -t [tagnameX,...,tagnameXn]
```

You may specify any number of new tags to be created provided that the resultant command length does not exceed the maximum of `4096 bytes`. You also need not worry if a tag already exists but you did not know it at the time. **The file system will skip the creation of tags that already exist and tell you if that happend.** You can call this command from anywhere, that is, it does not care about the current working directory.

Creating New Files

```
Arboreal >> new -f [filenameX(.ext),...,filenameX2(.ext)]
Arboreal >> new -f tagnameX/.../tagnameXn/filename(.ext)
```

There are two ways of creating a new file and they have a very important distinction between them:

- Version 1 allows you to create any number of files **within the current working directory ONLY**
- Version 2 allows you to create a **SINGLE file regardless of the current working directory however, you MUST specify the absolute "path" of the file.** *(In a future update we may implement relative paths for this as well)*

If a file could not be created (possibly because it already exists), an appropriate message will be returned instead of the newly created file's information.

"Delete" Commands

Deleting Tags

```
Arboreal >> delete -t [tagnameX,...,tagnameXn]
```

Delete any number of tags. **Tags will be deleted ONLY if they are empty.**

Deleting Files

```
Arboreal >> delete -f [filenameX(.ext),...,filenameXn(.ext)]
Arboreal >> delete -f tagnameX/.../tagnameXn/filename(.ext)
```

As with the `new` commands, there are two ways to delete a file:

- Version 1 will delete any number of files **provided that they exist within the current working directory.**
- Version 2 will delete a **single file regardless of current working directory, however, the file's absolute "path" MUST be provided.** *(In a future update we may implement relative paths for this as well)*

Although extensions on files are optional for these commands, you should be aware that:
`delete -f [BookReport.pdf]` will NOT delete `BookReport` `delete -f [BookReport]` will NOT delete `BookReport.pdf`

"Tag" Commands

```
Arboreal >> tag [filenameX(.ext),...,filenameXn(.ext)] +> [tagnameX,...,tagnameXn]
Arboreal >> tag tagnameX/.../tagnameXn/filename(.ext) +> [tagnameX,...,tagnameXn]
```

You may start noticing a pattern here of commands coming in pairs. This is completely natural and you are not going crazy. As with previous commands there are two ways to tag a file:

- Version 1 will **tag any number of files within the current working directory with any number of tags**. If the file or files have already been associated with a given tag then that tag is skipped.
- Version 2 will **tag a file regardless of the current working directory but the file's absolute path MUST be provided.**

(As always extensions on file names are optional but not providing them (if the file has one) will result in the expected file not being tagged or else an error stating that the file does not exist)

"Untag" Commands

```
Arboreal >> untag [filenameX(.ext),...,filenameXn(.ext)] -> [tagname1,...,tagnameXn]
Arboreal >> untag tagnameX/.../tagnameXn/filename(.ext) -> [tagnameX,...,tagnameXn]
```

- Version 1 will untag any number of files **provided that they are within the current working directory** and that they **are in fact tagged with the tags that you wish to be removed**.
- Version 2 will untag a **single file regardless of the current working directory,** provided that the file's **absolute path** is given and that the file is tagged with the tags you want to remove

(As always extensions on file names are optional but not providing them (if the file has one) will result in the expected file not being untagged or else an error stating that the file does not exist)

"Merge" Commands

Please note that these are still in production and not currently available

```
Arboreal >> merge tagnameX => tagnameY
Arboreal >> merge [tagnameX1,...,tagnameXn] => tagnameY
```

- Version 1 will merge `tagnameX` into `tagnameY`
- Version 2 will merge `tagnameX - tagnameXn` into `tagnameY`

"Open" Commands

```
Arboreal >> open filename(.ext)
Arboreal >> open tagnameX/.../tagnameXn/filename(.ext)
```

- Version 1 will open a file **provided that it exists within the current directory**
- Version 2 will open a file regardless of the current working directory **provided that the file exists and the file's absolute path is provided**

"Close" Commands

```
Arboreal >> close filename(.ext)
Arboreal >> close tagnameX/.../tagnameXn/filename(.ext)
```

- Version 1 will close a file **provided that it is currently open**
- Version 2 will close a file regardless of the current working directory **provided that the file is open and the file's absolute path is provided**

"Read" Commands

Please note that these are still in production and not currently available