README.pdf
============

Asst3.tgz - Will untar into ./Asst3 directory

Directory Structure after make:

```
./Asst3/Client                  : Root directory for client-side
archives/repositories
./Asst3/Client/WTF              : Client executable

./Asst3/Server                  : Root directory for server-side
archives/repositories
./Asst3/Server/WTFserver        : Server executable
```

Running the client and server:

The client and server executables are deployed in different subdirectories so that each one
has a separate archive/repository root directory.

To run the client, cd into the Client subdirectory under ./Asst3, and then run WTF.

To run the server, cd into the Server subdirectory under ./Asst3, and then run WTFserver.

Design Notes:

Client-Server Protocol - See Client-Server Protocol section below for details.

Server Multithreading -

We use the Pthread POSIX thread library to enable the server to handle multiple requests
simultaneously. When a WTF client connects to the main server socket, a new thread and server
socket is created. The thread handles the client's request and exits.

Server Project-level locking -

We use the Pthread library for mutex locking. The server maintains a list of project names,
where each project has a mutex created for it.

There is also a single mutex around the list itself, so that only a single thread can read the
list at a time, and may safely add a new project name and mutex. When a

```
client creates a new project,
        the thread will lock the list, then check the list and find no existing
entry and mutex, and will
        add a new one.  This prevents two clients who are both trying to create the
same project from creating
        two separate mutexs.  The project list mutex is released as soon as the
thread finds the specific
        project's mutex, or has created a new one.

        For an existing project, if two threads try to access the same project, the
first one to lock the
        project mutex list will proceed, while the other thread will block until the
first thread releases the
        mutex.

        All operations must be locked, even if they are read-only, because a
different thread may modify the
        project while it is being read.

File Hash Generation -

        We use the OpenSSL library to generate an SHA1 digest hash.


Client-Server Protocol:
------------------------
- We use a 2 byte code for the request type.

- For filenames, we send the length of the filename in bytes terminated with a ":".
Then we send that many
bytes for the name.

- For files, we send the length of the file, terminated with a ":", followed by the
file contents.
We use "\0" as a message terminator (<EOM>).  (This is safe because any \0
characters within a file are read
as part of the file based on the length of the file itself.  The EOM terminator will
only exist outside any
file data.)  If there are multiple files, the server will read each one using the
exact file length, and stop
reading when it reaches the terminator.


# Checkout
00<# bytes>:<projectName><EOM>
# Response from Server
00<# bytes>:<.manifest><# bytes>:<file name 1><# bytes>:<file 1>...<EOM>
01<Error message><EOM>

# Push
```

```
01<# bytes>:<Project Name><# bytes>:<.commit file><# bytes>:<file name 1><#
bytes>:<file 1>...<EOM>
# Response from Server
00<EOM>
01<Error message><EOM>

# Update
02<# bytes>:<Project Name><EOM>
# Response from Server
00<# bytes>:<.manifest><EOM>
01<Error message><EOM>

# Upgrade
03<# bytes>:<projectName><# bytes>:<file name 1><# bytes>:<file name 2>...<EOM>
# Response from Server
00<# bytes>:<file name 1><# bytes>:<file1>...<EOM>
01<Error message><EOM>

# Create
04<# bytes>:<projectName><EOM>
# Response from Server
00<# bytes>:<.manifest><EOM>
01<Error message><EOM>

# Destroy
05<# bytes>:<projectName><EOM>
# Response from Server
00<EOM>
01<Error message><EOM>

# Current Version
06<# bytes>:<projectName><EOM>
# Response from Server
00<# bytes>:<.manifest><EOM>
01<Error message><EOM>

# History
07<# bytes>:<projectName><EOM>
# Response from Server
00<# bytes>:<.history><EOM>
01<Error message><EOM>

# Rollback
08<# bytes>:<projectName><versionNumber><EOM>
# Response from Server
00<EOM>
01<Error message><EOM>

# Commit
09<# bytes>:<Project Name><# bytes>:<.commit file><EOM>
```

```
# Response from Server
00<EOM>
01<Error message><EOM>
```