# kron Analysis

## File structure:

To record the information of every revision including the changesets and manifests efficiently, we design the file structure as below:

```
.kron
├── index
├── objects
|   └──<snapshots of file>
├── changesets
|   └──<changeset for each revision>
├── manifests
|   └──<manifest for each revision>
├── rev (the entire revision tree)
├── heads
└── stage
```

**Index**: This file records the filenames and some information in the tracking list.

**Objects**: This folder is to store all files data committed to repository in each revision. It will be divided into several sub-folders by the first two characters of filename in the objects folder.

**Changesets**: This folder is to store all changesets of the repository. Each changeset of a revision is stored by a single file in the changeset folder.

**Manifests**: This folder is to store all manifest of the repository. Each manifest of a revision is stored by a single file in the manifest folder.

**Rev**: This file is to store the entire revision tree including the commit history and each revision number(hash) .

**Stage**: This file records which files are in stage area waiting to be commit.

**Heads**： This file records the latest revision number of each branch.

## Modules:

1. Repository:

This module will give some interface to operate the repository.

2. Revision :

   This module will create and update the revision tree to record information of every revision.

3. Changeset:

   This module will record which files are modified in this revision.

4. Manifest:

   This module will operate the files in manifest folder.

# End to end test:

1. Test `init`

   ```
   $ kron init
   ```

   Expect result: All the files in .kron will be initialized.

2. Test `add`

   ```
   $ kron add A.txt B.txt
   ```

   Expect result: Stage file will be changed as below:

   > /path/to/repo/.kron/stage
   > + A.txt
   > + B.txt

   `+` means add. And index file will add a record as below:

   > .kron/idx
   > /path/to/repo/A.txt (hash of A.txt)
   > /path/to/repo/A.txt (hash of B.txt)

3. Test `diff`

   If some files are added, deleted or modified compared to the last revision , then use this command the expect result:

   ```
   $ kron diff
    + A.txt
    + B.txt
   ```

4. Test `remove`

```
$ kron remove --index B.txt
```

Expect result: the records of B.txt will be removed. stage file will be changed as below:

> /path/to/repo/.kron/stage
>
> + A.txt

`+` means add. And index file will add a record as below:

> .kron/idx
>
> /path/to/repo/A.txt (hash of A.txt)

5. Test `status`

```
$ kron status
```

Expect result:

```
$ kron status
On branch master
Changes to be committed:

    new file:   A.txt
```

If you create a file, C.txt, but forget to add it to the tracking list, then the result will be:

```
$ kron status
On branch master
Changes to be committed:

    new file:   A.txt

untracked files:C.txt
```

6. Test `commit`

```
$ kron commit
```

Expect result: File structure:

```
.kron
├── changesets
|   └── aaf44dbe3ca6492f7272e8b207c912929e0aa170
├── config
├── index
├── manifests
|   └── aaf44dbe3ca6492f7272e8b207c912929e0aa170
├── objects
|   └── ac
|       └── ac882f6e899f616e9cf445d2613b1fe33d1db2d6
├── rev
├── heads
└── stage
```

```
/path/to/repo/.kron/heads

master aaf44dbe3ca6492f7272e8b207c912929e0aa170
head aaf44dbe3ca6492f7272e8b207c912929e0aa170
```

```
/path/to/repo/.kron/objects/ac/ac882f6e899f616e9cf445d2613b1fe33d1db2d6

(the same content of a.txt )
```

```
/path/to/repo/.kron/manifests/aaf44dbe3ca6492f7272e8b207c912929e0aa170

/path/to/repo/A.txt ac882f6e899f616e9cf445d2613b1fe33d1db2d6
```

```
/path/to/repo/.kron/changesets/aaf44dbe3ca6492f7272e8b207c912929e0aa170

user:xxxxxx time:xx-xx-xxxx
add:filenames
delete:filenames
modify:filenames
commit-message:"xxxxxxxx"
```

There are two files created in manifests folder and changesets folder seperately, they have the same name of the hash of this revision. And there is also a file created in objects folder. The content in stage will be clear. heads will be created to record the hash of head .

7. Test `log`

```
$ kron log
```

Expect result:

```
$ kron log
commit aaf44dbe3ca6492f7272e8b207c912929e0aa170 (HEAD -> master)
Author: yuzhang <zhangyu@zhangyus-MBP.rochester.rr.com>
Date:   Tue Oct 9 21:27:32 2018 -0400

    'commit message'
```

8. Test `cat`

```
$kron cat A.txt aaf44dbe3ca6492f7272e8b207c912929e0aa170
```

Expect result:

```
$kron cat A.txt aaf44dbe3ca6492f7272e8b207c912929e0aa170
/path/to/repo/A.txt

test
modify
modify2
```

9. Test `head`

```
$ kron heads
```

Expect result:

```
$ kron heads
heads on master:aaf44dbe3ca6492f7272e8b207c912929e0aa170
```

10. Test `cheakout`

```
$ kron checkout <hash of revision>
```

Expect result: The working directory will return to the given revision. The heads file will change base on manifest of given revision. The information of head will also be changed

11. Test `pull`

```
$ kron pull <name of repository>
```

Expect result: The working directory will not be changed. The objects folder, manifests folder and changeset folder will be merged, with files which have the same filename(hash) be overwrite. The rev file of two repositories which records the revision tree will be merged based on the same parent.

12. Test `merge`

```
$ kron merge <name of branch>
```

Expect result: Check if there are conflicts between the manifest of the latest revision of two branches. If there is no conflict, then find all the files listed in the two manifests, clear the working directory and recover them to the working directory. The head recorded in heads file will be changed to two hash of the head revision of two branches we merge. The merge result will not be recorded until you commit.

13. Test `clone`

```
$ kron clone <name of repository>
```

Expect result: All file structure of target repository including the working directory will be copied to local.

14. Test `push`

```
$ kron push <name of repository>
```

Expect result: The same effect as the target repository clone a local repository