Kron Architecture

This document contains a description of modules and interfaces, and the USE hierarchy between these modules.

Modules

#1 Repository

This is an abstraction of the entire version-controlled work directory. It implements command such as `commit` , `pull` and `checkout` which acts as a boundary which user interacts with. This structure keeps an instance of *Revision* (the latest committed record) and handles several operations indirectly by invoking that instance's methods.

#2 Revision

A core module to maintain the changes and structure of versioned files by synchronizing records in folder `.kron/`.

#3 Changeset

A submodule of *Revision* to keep track of commit information and file modifications.

#4 Manifest

A submodule of *Revision* to store the work directory structure for a specific revision. It keeps a record of origin files' metadata and maps them to specific locations for future extraction and recovery.

USE Hierarchy

> The first level modules are module #1 and #2, and the second are module #3 and #4.
>
> #1 calls #2, and #2 uses its built-in object #3 and #4, which shows in **Figure 1**:

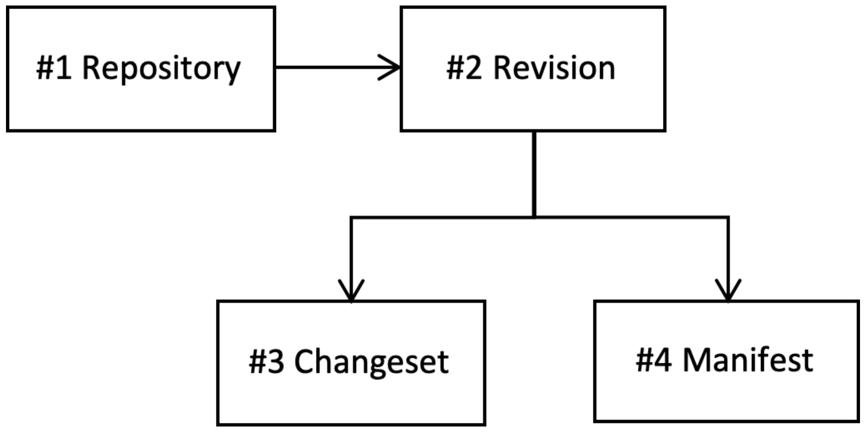

Figure 1: The USE hierarchy between core modules.

Method Signatures

#1 Module `Repository`

- ***Repo Initialization:***

  ```
  init(repo_dir='./')
  ```

  This method takes an optional parameter `repo_path` to initialize a version-controlled repository.

- ***Add file(s) for version control:***

  ```
  add(file_path, ...)
  ```

  Add file(s) to the tracking list (stage) of the current branch.

- ***Remove file(s):***

  ```
  remove(force=false, file_path, ...) -> Array[Boolean]
  ```

Untrack specific file(s) from file `index` and delete the original file in the work directory if `force` is set to true. It returns a boolean array of execute result of each file.

- ***Check stage status:***

  ```
  status -> set
  ```

  Returns a set of status (stage files, account, last commit, etc.) for current repo.

  This method will invoke `Revision.check_stage` to update changes from the working directory.

- ***Add branch:***

  ```
  add_branch(branch_name) -> Boolean
  ```

  Add a new branch.

- ***List branches:***

  ```
  get_branches -> Array[String]
  ```

  Returns a list of name of each branch.

- ***List heads:***

  ```
  heads -> Array
  ```

  Returns a list of head revisions of each branch.

- ***Get current revision:***

  ```
  revision -> Revision
  ```

  Returns the current revision.

- ***Load a revision:***

```
load(rev=HEAD) -> Self
```

Loads a specific revision for current repo. Files in the working directory will be changed after this call.

- ***Check out a specific revision:***

  ```
  checkout(branch=master, revision=branch.HEAD) -> Revision
  ```

  Returns a specific revision of a specific branch, the default branch is called 'master', the default revision is the HEAD of that branch.

- ***Retrieve the text of a certain revision of a certain file:***

  ```
  cat(file_path, rev=HEAD) -> File
  ```

  This method takes 2 parameters: the file to inspect and the revision id. It then retrospects a history version of a specific file.

- ***Clone a repo:***

  ```
  clone(url) -> Self/nil
  ```

  Clone from an existed repository.

  The parameter `url` can either be a `repo_dir` or a `remote_url` , the module can automatically find the repo and download the compressed repository file into the current folder.

- ***Push changes:***

  ```
  push(repo_dir) -> Boolean
  ```

  Push changes to a specific repository and returns `True` if repo successfully pushed.

  This method takes a repo_dir as a parameter, pushing the `changeset` and `manifest` of the current repo to target repo.

- ***Pull changes:***

  ```
  add(repo_dir) -> Boolean
  ```

  Pull changes from a specific repository and returns `True` if that repo has successfully pulled.

  This method takes a repo_dir as a parameter, finds the common ancestor from `changeset` and `manifest` of current repo with that repository's HEAD, then update the `rev` graph of current repo.

  This method uses `self.revision`.

- ***Commit a revision:***

  ```
  commit -> Self
  ```

  Commit changes, synchronize tracking status and update the current repository.

  This method uses `self.revision`.

- ***Merge two revisions:***

  ```
  merge(rev)
  ```

  This method takes a revision's id as a parameter, then merge it with the current revision of current repo. This method will find the common ancestor of two repositories i.e. their common ancestor of changelogs and manifests, then merge all the changes on the base of their ancestor revision.

  This method uses `self.revision`.

- ***Grab a changeset:***

  ```
  changeset(rev=HEAD) -> Set
  ```

  Returns the changeset of a specified revision.

- ***Grab a manifest:***

  ```
  manifest(rev=HEAD) -> Array[Manifest]
  ```

Returns manifest records of a specified revision.

#2 Module `Revision`

- ***Get global identifier:***

  ```
  id -> String
  ```

  Returns a SHA-1(TBD) hash of current revision to identify current revision records (manifest and changeset).

- ***Get node identifier:***

  ```
  node_id
  ```

  Returns the row_index of current revision in the `.kron/rev` file (to identify the current node in the DAG graph of revisions).

- ***Get changeset:***

  ```
  changeset -> Changeset
  ```

  Returns a changeset of current revision.

- ***Get manifest:***

  ```
  manifest -> Manifest
  ```

  Returns a manifest records of current revision.

- ***Load changeset:***

  ```
  load_changeset(create_if_null=true) -> Self
  ```

  Loads changeset from file `.kron/changeset/current_changeset`.

- ***Load manifest:***

  ```
  load_manifest(create_if_null=true) -> Self
  ```

  Loads manifest from file `.kron/manifest/current_manifest`.

- ***Sync changeset:***

  ```
  sync_changeset -> Self
  ```

  Save changeset to file `.kron/changeset/current_changeset` if content changed.

- ***Sync manifest:***

  ```
  sync_manifest -> Self
  ```

  Save manifest to file `.kron/manifest/current_manifest` if records changed.

- ***Get parent revisions:***

  ```
  parents -> Array[Revision]
  ```

  Returns parents of current revision.

- ***Set parent revisions:***

  ```
  parents(rev1, rev2) -> Self
  ```

  This method takes two parameters `rev1` and `rev2` as the two revision as parents of current revision.

- ***Get ancestor revision:***

  ```
  ancestor(rev1, rev2) -> Revision
  ```

Return an ancestor revision (least common ancestor) of revision1 and revision2.

- ***Merge revisions:***

```
merge(base=ancestor(self, rev), rev) -> Revision
```

Merge a revision into current revision from the base node and returns a new revision of that.

- ***Add child revision:***

```
add_child(rev) -> Revision
```

This method takes a parameter of a `Revision` object and returns that object with filled attributes (such as id, node_id, parents, etc.).

#3 Module `Changeset`

- ***Enumerate each change records:***

```
each { |changed_item| block }
```

- ***Create a changeset:***

```
Changeset.builder.added_files(Array)
                 .changed_files(Array)
                 .removed_files(Array)
                 .commit_message(String)
                 .utc_time(Datetime)
                 .account(String)
                 .new

add(added_files=[], changed_files=[], removed_files=[], message, time=Now,
account)
```

#4 Module `Manifest`

- ***Enumerate a manifest:***

  ```
  each { |mf| block }
  ```

  Enumerate each file-path mapping record in a manifest file.

- ***Add a mapping record:***

  ```
  Manifest.add_record(
      ManifestItem.builder
                  .name(String)
                  .dir(Boolean=False)
                  .path(String)
                  .format(String)
                  .compressed(Boolean=False)
                  .ctime(Datetime)
                  .mtime(Datetime)
                  .checksum(String)
                  .new)

  add_item(ManifestItem)
  ```

  > A `Manifest` consists of multiple `ManifestItem` objects.

  Dir - A boolean tag for a directory record, corresponding to the parent node of a `tree object` in Git.

  Compressed - In order to save disk space, we decided to compress files for our latest revision records. However, instead of compressing all the contents, we zip only a part of them by skipping files with formats like JPEG, TAR, ZIP, etc.