Version Control Polices and Procedures

1. General Use

The directory ~gfisher/projects/support/bin has the following executable programs that support version control:

```
chkin -- check in one or more files
chkout -- check out one or more files, including an entire project
steal -- check out a file that belongs to someone else
unsteal -- allow a stolen file to be checked back in

assign -- assign a file to an owner
reassign -- reassign a file to a new owner
release -- release one or more files or the whole project
purge -- remove old versions no longer needed
prune -- prune branches from one or more files
strict -- put a project in strict mode, forcing test before chkin or release
```

The first four commands are available to all project members. The last six are for project librarians only.

The `chkin` command is limited to the owner of a file. It archives a new version of the given files. With no arguments, it checks in all files that have changed since the last check in, and outputs a list of the files that were checked in. With the the argument "-t" in runs in "test mode", outputting messages but not actually doing the check ins. "-t" is useful for checking to see which files have changed since the last check in.

The `chkout` command is available to all project members. When chkout is run by the owner of a file, a copy of the file is made. When chkout is run by a non-owner of a file, a symbolic link to the most recently checked in version of the file is made. With no arguments, `chkout` checks all of the files of a project.

The `steal` command allows a non-owner to check out a file by copying rather than linking. The file is marked as stolen in the project log, so that a subsequent attempt by the owner to check the file in will prevented, with a message indicating who has stolen the file. At this point, the owner and stealer must reconcile any separate changes that have been made to the file and the stealer must use the `unsteal` command to allow the file to be checked in by the owner.

The `assign` command assigns a user to be the owner of one or more files. A file can only have a single owner. `reassign` changes the owner of one or more files.

The `purge` command permanently removes old versions from the archive. It takes one or more file names and a version number as defined below. All versions at and before the given number will be purged from the archive.

The `prune` command prunes alternative branches, which are created as described in the section on Reversion and Branching. `prune` takes the letter of the branch to prune or the name "main" which refers to the main branch. With no letter, it prunes the branch with the alphabetically last letter. When a branch is pruned, all alphabetically later versions are promoted up one letter, with the main branch treated as alphabetically earliest. `prune` can be given two branch letters, in which case it doesn't really prune, but rather swaps branches. This is primarily useful to promote some branch other than 'a' to main. If there are no branches defined, the `prune` command does nothing.

The `release` command releases files from a working directory to a release directory. The first argument is the release number, which consists of the first five parts of a the general version number as defined below in the section on Version Numbering. If the version number argument is missing, the default release is a new platform independent, alpha-level minor update. With no file name arguments, the entire project is released.

2. Reversion and Branching

`chkout` can be given a version number argument to check out a file version earlier than the current version. The version number can be absolute, in the form of x.y.z, or relative, in the form of -n for n = 1 to the total number of checked in versions.

Whenever an older version of a file is checked out, a *branch* in the version history is created. Version branches are designated with letters rather than numbers, as described in the section on Version Numbering below.

This system does not allow the kind of branching that is typical in other VC systems, where two or more branch paths can be created for a single file at any point. The style of this system is more conservative, viewing branching as an exceptional rather than normal condition. That is a branch is only created when an older version of a file is checked out.

Stealing may be thought as a weak form of branching, however it does not directly affect the archive in that no new versions are generated.

3. Version Numbering

[Ed. Note: Probable update is to make platform name and release level appended bookmarks, not embedded in the actual vc number. So, the version number will look like this

```
M.m.u[.br].c.b.t.s.e p/l/...[additional bookmarks]
```

The version number for any file is of the following form:

```
M.m.u[p/][l]c[br].b.t.s.e
```

where

`M` = major release
`m` = minor release
`u` = minor release update
`p` = optional platform name
`l` = optional release level of alpha or beta
`br` = optional branch
`c` = check in
`b` = build
`t` = tool session
`s` = save
`e` = edit

For example, `4.1.3solaris/alpha.8.4.15.6.87`.

The version number counter runs like an odometer, in that every time an upper number is incremented, all numbers below it are reset to 0.

Version branching only occurs at the check-in level, and only when an older version of a file is checked out, as described above in the section on Reversion and Branching.

4. Strict Mode

The `strict` command puts the project in strict mode, which means that the system will automatically run tests on any files when a `chkin` or `release` command is run. Details of how tests are run are forthcoming.

5. Other Functional Details

There needs to be some kind of companion file that resides with or near each versioned file, say in an ART directory containing `.art` files. That dir will have a file with the version information, among other things that other Inferno tools may need.

As each tool wields a file, it updates the low end of the version number. When a file is checked in and released, the VC tool updates the upper end of the version number. There remain questions about how many copies of the actual and companion `.art` files there need to be at the check-in level. There are also remaining questions about the role of `.art` files, at the user-visible level and/or at the implementation level.