Gitolite Repository Hosting

The LSST repositories are managed with **gitolite**. If you're a new user, read here to get started.

Gitolite Remote Commands

A number of useful commands can be triggered by both admins and devs, by ssh-ing to git@git.lsstcorp.org. These are gitolite's admin defined commands. The following alias is also useful (and used in the examples below):

```
alias gitolite="ssh git@git.lsstcorp.org"   # bash users
alias gitolite "ssh git@git.lsstcorp.org"   # (t)csh users
```

Use the *info* command to see which repositories you are allowed to access. Use the *expand* command to show the full list of repositories to which this expands:

```
gitolite info    # see which repositories you can access
gitolite expand  # see all repositories to which these expand
```

Note that the patterns shown in the output are regular expressions, not shell globs (i.e., '.' character is a stand-in for 'any character', and not a dot as it is with shell globs).

Creating a Repository

Repositories are actually auto-created when you try to *clone*, *push*, or perform any other operation on a nonexistent repository. For example:

```
mkdir newthing
cd newthing
git init
#  Write code, then add (all) the new files to your staging area:
git add .
#  Commit the files to the local branch
git commit -m "New awesome package"
#  Now add a new remote repository for your package:
git remote add origin git@git.lsstcorp.org:contrib/newthing.git
#  Push the files to the remote repository; -u adds upstream tracking
git push --all -u
#  Update the tags too:
git push --tags
```

Creating a repository in the **LSST** hiearchy requires a special procedure.

Deleting a Repository

To remove a repository (assuming you are its owner), use the *trash* command:

```
gitolite trash contrib/obsolete_thing.git
```

This command doesn't actually delete the repository; it only moves it to `trash`. Use *list-trash* to see what's in the trash, and *restore*, to restore a deleted repository from the trash. To permanently delete a repository, someone with the git account password must do the following:

```
ssh to git@git.lsstcorp.org
cd ~/repositories/deleted
rm -rf newthing.git
```

This is a safety feature: it should be very hard to permanently delete anything from a repository.

Forking a Repository

Forking is essentially making a server-side repository copy (a clone), allowing you to clone an existing repository (e.g., for experimentation). For example:

```
gitolite fork LSST/DMS/afw personal/mjuric/afw
git clone git@git.lsstcorp.org:personal/mjuric/afw
```

This now gives me a full clone of afw, with full rights to all of its tags/branches/etc. Note: the server-side clone is done in a ***very*** space-efficient way (hardlinking is used where possible). If you want to fork the repository to a remote site (e.g., for control of the stability of the stack while retaining the ability to exchange commits), see this.

Renaming or Moving a Repository

To re-name or move a repository, do:

```
gitolite mv <from_name> <to_name>
```

Unlike UNIX ***mv***, `<to_name>` cannot be a directory: it must be a fully-qualified repository name. For example:

```
gitolite mv contrib/mything.git contrib/bettername.git
```

Note that the `.git` extension is optional in the above command.

Admin-Only Operations

Creating a Repository in LSST/ Hierarchy

One cannot directly create (or delete, but you can trash) repositories in `LSST/`, but if you're a member of *@admins*, you can move an existing user repository there using:

```
gitolite sudo root mv <from_repo> <to_repo>
```

Since repositories are auto-created if they don't exist (see above), allowing even a small subset of *@admins* to create them in the `LSST/` hierarchy may lead to lots of spurious repos due to typos. Because of that, only the user '`root`' is allowed to create (or delete) repositories in `LSST/`, and user '`root`' can only be accessed using the '`sudo`' command. This adds an additional layer of safety.

Deleting a Repository (that you don't own)

To remove a repository (if you are not the owner), check who is the owner, and run the *`trash`* command as the owner:

```
gitolite expand contrib/obsolete_thing.git
gitolite sudo <owner> trash contrib/obsolete_thing.git
```

SSH-ing into the git Account

Use a password to get shell access (as keys will redirect to gitolite). To force SSH to skip public key auth, do:

```
ssh -o PubkeyAuthentication=no git@git.lsstcorp.org
```

Managing LSST gitolite Users and Permissions

User and permission management is done via configuration files in the standard gitolite-admin repository. You have to be a member of @admins gitolite group to access this repository. Membership consists of roughly one person per DM site. For the exact list, contact lsst-admin@...

For user management activities, first clone **gitolite-admin** to your local machine:

```
git clone git@git.lsstcorp.org:gitolite-admin
```

Adding or removing users

Add the user's SSH public key to a file named keydir/username@0.pub in the gitolite-admin repository. For example, to add user mjuric, add mjuric's public key into:

`keydir/mjuric@0.pub`

If the user has more than one key, add as many as necessary by changing the numeric suffix following the @ sign (e.g., keydir/mjuric@1.pub, etc.).

Next, add the user to conf/devs.conf, to the @devs group. For example:

`@devs = mjuric`

Commit the changes, and push them upstream to make the changes effective:

```
git commit -a
git push
```

To remove users, simply remove the public keys and their entries from `conf/devs.conf` file, then *`commit`* and *`push`*.

Removing Accident From a Repository

Large files (usually, test data) have on occasion been added accidentally to a repository. Below is a recipe for *`@admins`* to remove them.

> **NOTE: This is history rewriting and should be done only after consulting mjuric!**

- `git clone`, plus don't forget to create another clone for safety, comparison, etc.
- Then do:

  ```
  git filter-branch -f --index-filter \ 'git rm --force --cached --ignore-unmatch <pathToFile>' \ -- --all
  ```

  In our case the <pathToFile> looked like `tests/case02/data/Object.txt, tests/case02/data/Source.txt` etc. Repeat for every file you are removing.

- Remove the files push the result (using the force):

  ```
  rm -Rf .git/refs/original && \ git reflog expire --expire=now --all && \ git gc --aggressive && \ git prune
  git push --force
  ```

- Checkout the branch where the offending files were committed and "`git push --force`" that branch

Backup

To back up all of the repositories simply back up anything and everything in `~git`. Note that `~git/repositories` is a symlink to `/lsst_ibrix/gitolite/repositories`, that should be backed up as well.