Auth/SSO systems

A major part of Lix infrastructure is the authentication/SSO systems. Here, you can find information about how to run them.

- Changing names, emails, etc
- How accounts work
- How do permissions work?
- Assigning Groups
- Tutorial: adding auto mapping of forgejo groups

Changing names, emails, etc

The Lix project endeavours to not deadname people, because we believe in human decency. However, some of our software has other ideas. This page documents the workarounds to manually fix profile updates that don't get conducted because various software is busted.

Intended design

Ideally, contributors should be able to go to https://identity.lix.systems and change their usernames, display names and emails and relog every service, and then every service will have correct names and emails.

wiki.lix.systems

The wiki does not update emails when they are changed via OIDC. Furthermore, users can't change them themselves. Why do they do this, we will never know; OIDC has persistent UUIDs, they have no reason to do this.

To fix a user's email manually, go to https://wiki.lix.systems/settings/users, and select the user in question and edit them.

The wiki will also not change fullnames automatically, which is also broken, but users can simply change them. It does not seem to use usernames at all.

git.lix.systems

!! Currently this is broken and we cannot change forgejo usernames at all: https://git.lix.systems/lix-project/web-services/issues/93. The workaround here is to clear the username field when changing to a local account.

Forgejo blocks username changes for accounts with external sign-in for no reason. These have to be fixed by an administrator. Go to https://git.lix.systems/admin/users, click the edit icon next to the user in question, then set the Authentication Source to Local, fix the username, then press Update User Account. Next, set the Authentication Source back to Lix.

Users are able to change emails themselves by adding a new email then deleting the old one. They can also be changed by administrators in the same page as above.

OIDC has persistent UUIDs, there is no reason for Forgejo to do this.

Forgejo does not update names or emails from Keycloak after initial login, which is broken as well.

gerrit.lix.systems

Gerrit will break accounts rendering them incapable of logging in if they change username. Changing email and display name works as expected. It appears that Gerrit wants to believe that usernames are not possible to change, which is a skill issue, because they *have* numeric IDs.

Extremely untested scuffed-looking db hacking procedure:

https://wikitech.wikimedia.org/wiki/SRE/LDAP/Renaming_users/Gerrit

pad.lix.systems

We think this one works properly last time we checked. It seems to just replace the profile on each login.

How accounts work

Lix has one source of truth for authentication: Keycloak (identity.lix.systems). Most services are bound to Keycloak for authentication via OAuth2, although it supports SAML as well.

GitHub vs Local accounts

GitHub accounts are used at Lix for two reasons:

- Ease of login and onboarding
- Ban/allow list management

We don't really care if people have the same username or other information on Lix as they have on GitHub. We don't care about whether people have first/last names on our Keycloak or if they are using pseudonyms.

Allow/ban listing

There is an allow-list and a ban list maintained at: https://git.lix.systems/lix-project/access-control (private repo, available only to Lix core team). To add people to a list, use `./add.sh list.txt gh-username`. Once a list change is pushed, it can take up to five minutes for the change to take effect, as this is currently running on a 5m cron job.

In short, the process for adding a user to the ban or allow list is:

1. Make sure you have the latest version of the repo (i.e. `git pull`) and the github `gh` command is installed.
2. Run `./add.sh <relevant-list-file> <github username>`.
3. Commit and push the change.
4. The ACL change will apply automatically within five minutes.

Be warned -- the allow-list method of access control is temporary / established for the beta period.

Our allow/ban listing is done by GitHub ID, using keycloak-allowban-plugin, a custom Keycloak plugin that reads text files with allow/ban lists. The GitHub ID is put into a user profile attribute, which prevents ban-evasion via account unlinking since it will stick across unlinking.

Known weirdness with the allow/ban list plugin

If a user tries to log in via GitHub and they are not allowed by the plugin, the account is created anyway, it is simply not usable. This is a known issue; putting the plugin in the registration flow caused half-registered users, so it is only in the post-gh-login flow and the normal login flow (to

catch unlinked banned accounts).

Local accounts

The Lix core team should have local accounts (linking to GitHub is OK), strongly preferably with 2FA. Other people can be given local accounts if they are trustworthy and prefer to have local accounts (since the usual ban process doesn't work on them; though it is not hard to ban them, just disable the account).

Note that GitHub backed accounts can be turned *mostly* into local accounts by the user simply setting up local auth and unlinking the GitHub account (though the GitHub ID will intentionally persist in properties so this doesn't degrade our bans story).

We would *prefer* for everyone to use WebAuthn for local accounts, but this is often not possible and passwords are OK as long as they're just put in a password manager.

To create a local account, get the following info:

- Username
- Email (which will appear publicly on Gerrit and must be deliverable)
- WebAuthn ok?

Then create an account on https://identity.lix.systems/admin with the provided details. On the account's page, go to Credentials, select Credential Reset, then if WebAuthn is ok for the person, set "WebAuthn Register Passwordless" in the actions (otherwise just password reset) and send it.

Removing last names for people

Due to Keycloak being a silly little thing, we need to use "declarative user profiles" to allow not setting last names. For now, Lix core team members with necessary access will have to remove them manually on request.

This would be fixed by updating Keycloak to 24 on lix.systems and setting up declarative user profiles: https://git.lix.systems/lix-project/web-services/issues/64

How to ban someone

If a user has violated our community norms and needs to have their access to our infrastructure removed, follow the following steps:

1. Add them to the banned users list on https://git.lix.systems/lix-project/access-control and push the changes.
2. Go to https://identity.lix.systems/admin and disable their account for good measure.

3. Ban them from Matrix: FIXME
4. (if you really don't like what's going on) invalidate all sessions:
    1. `ssh root@git.lix.systems -- mysql -D forgejo -e 'delete from session;'`
    2. `ssh -p 2022 youruser@gerrit.lix.systems gerrit flush-caches --cache web_sessions`
    3. FIXME: bookstack

How do permissions work?

In an ideal world, all permissions are managed directly in Keycloak and propagated down to downstream systems automatically. We mostly live in that world. We would also like more parts of profiles to propagate from Keycloak into downstream systems (see changing names document).

First, let's enumerate the access that we *have available* to grant.

Available access

Roles that exist in Keycloak

"sticky" is referring to whether later-removed permissions get stuck in the downstream system if they are removed upstream

- Wiki roles (not sticky)
  - Editor
  - ModerationBook
  - Admin
- Buildbot (not sticky)
  - can-perform-mutations (currently just a gate on all access)
- Forgejo (administrator is probably sticky, however others are not)
  - Administrator permissions
  - lix-project/nixos issue triage team
  - lix-project owners team
- Keycloak (not sticky)
  - lix-project-user-admin realm role grants administration on keycloak
- Grafana (?)
  - Admin
  - Editor
- Pad (not sticky)
  - access
- Mattermost (not sticky?)
  - access

Roles that we wish existed in Keycloak

These can't happen due to current technical limitations.

- Gerrit: https://git.lix.systems/lix-project/web-services/issues/100
  - Lix team

    - Possibly other rights like tvix-fork team or others?

Structure of access

Keycloak appears to want its structure to work like:

Group -> Composite Realm Role -> Client Role

We don't have that many groups or client roles to assign to make composite realm roles make much sense at the time of this writing.

When creating new clients, make their roles *client roles*, which we can then assign to other objects inside Keycloak so we can do role-based access control at a later time without having to mess with the services.

Groups

- lix core team -> grants administrative permission across a lot of infrastructure:
    - Wiki
        - ModerationBook
        - Admin
        - Editor
    - Forgejo
        - Administrator
        - lix-project owners team
    - Grafana
        - Editor
        - Admin
    - Mattermost
        - access
    - Pad
        - access
    - Buildbot
        - can-perform-mutations
- Lix community team -> grants administrative permission across a lot of infrastructure
    - Wiki
        - ModerationBook
        - Editor
    - Forgejo
        - Administrator
        - community-team
    - Mattermost TODO should probably get access, but currently doesn't (?)
    - Pad
        - access

- trusted-contributors
  - Forgejo
    - lix-project/nixos issue triage team
  - Wiki
    - Editor
  - Buildbot
    - can-perform-mutations

Policy on who goes in groups

- lix core team: *only* members of the Lix core team (this group will probably be rearranged later on)
- lix community team: *only* members of the Lix community team.
- trusted-contributors: anyone who is reasonably competent and trustworthy: these are basically the rights we hand to every single package maintainer in nixpkgs, and we want to do similarly in Lix.
  Especially assign this if someone should be able to have issues assigned to them (since it is a prerequisite); this means this role should be handed to anyone who is going to contribute code so we can use the assignment feature of Forgejo to say who is taking an issue.
  - /private beta: this group has been made to indicate who was in trusted-contributors already due to our simply dumping everyone from the private beta into it

Assigning Groups

See How do permissions work? for implementation details.

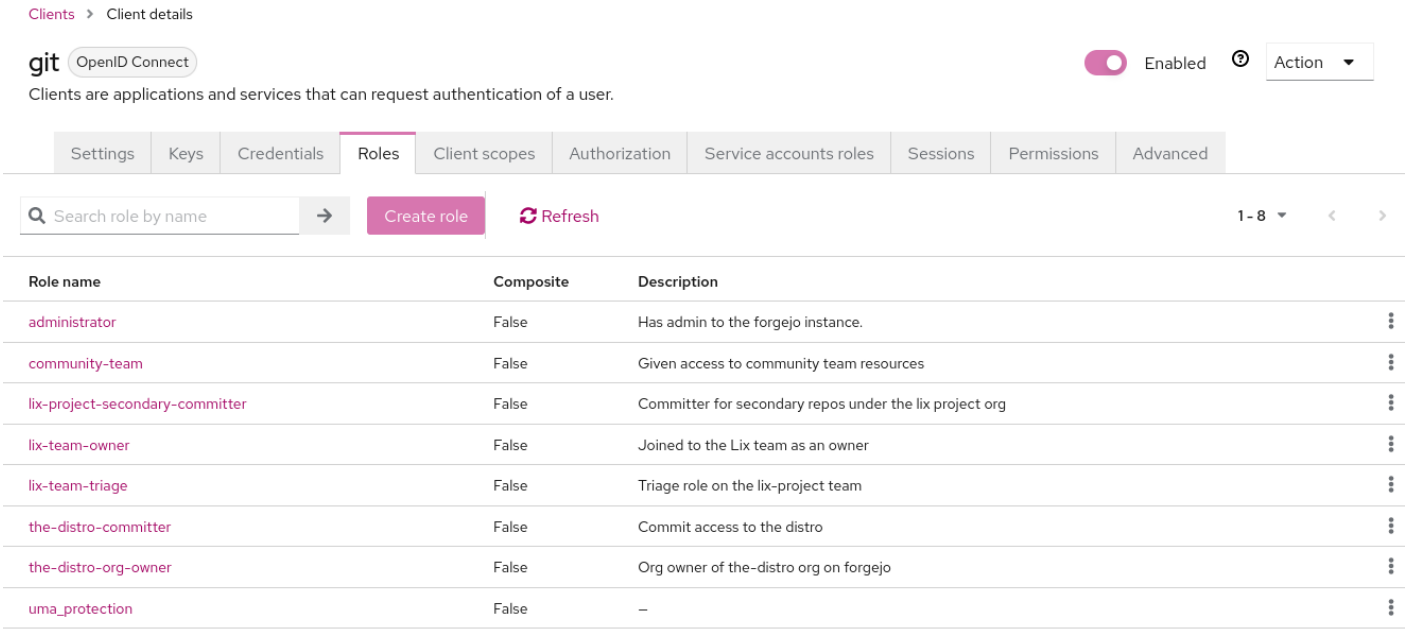
tldr;

- Go to the admin console (no trailing slash)
- Go to Groups -> YourGroupHere -> Members
- Add/Remove members as needed

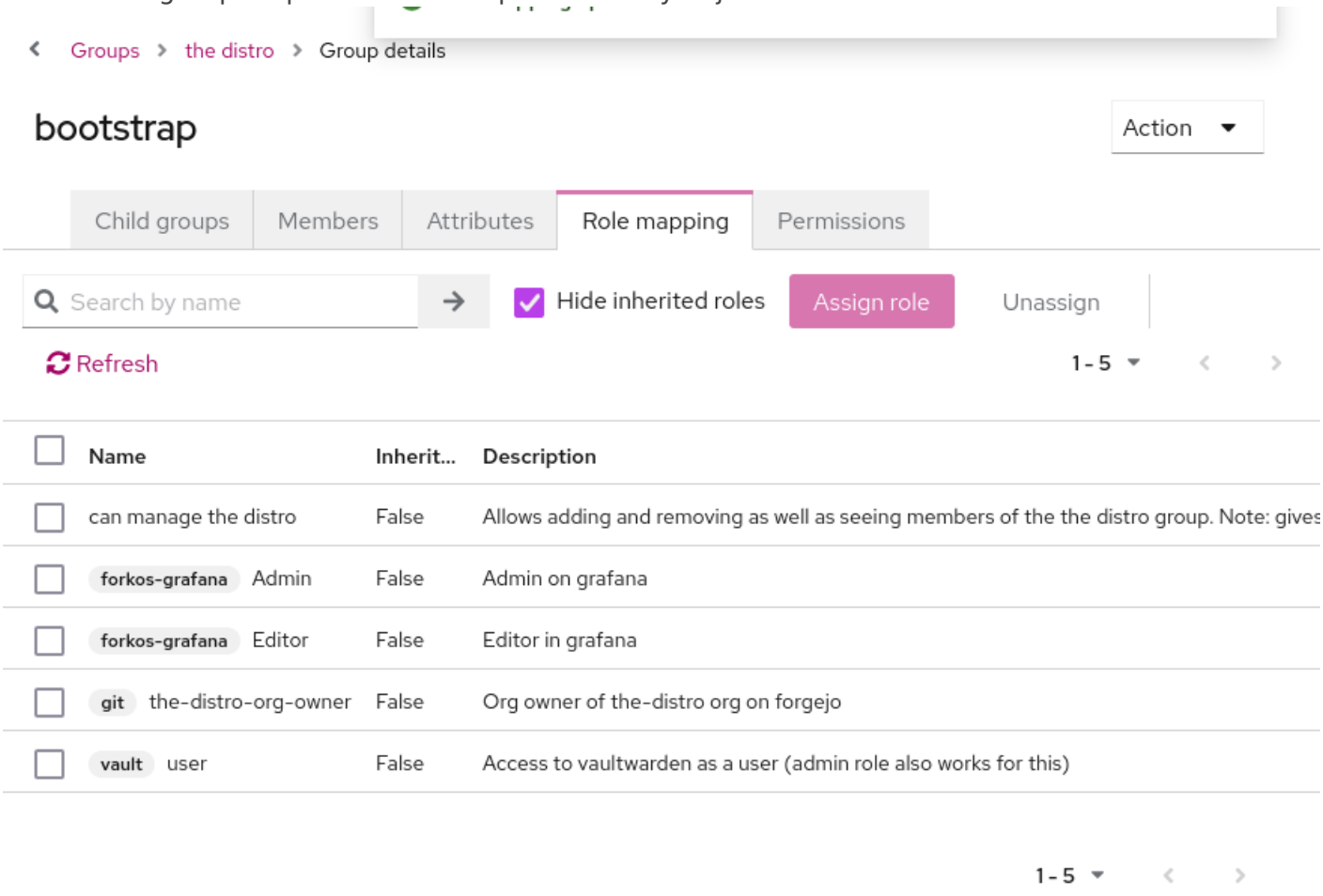
Note: most permissions only update after logging out and back into the appropriate application.

Tutorial: adding auto mapping of forgejo groups

Create a role on the Keycloak client:



Go into the group in question and map it the role you just made:



Add a json snippet to map the role in the incoming tokens to the appropriate team on the org:

```
{"the-distro-committer": {"the-distro": ["committers"]}, "the-distro-org-owner": {"the-distro": ["owners"]}}
```

It needs to be added to: https://git.lix.systems/admin/auths/1