Repo command reference

In this document

Repo usage takes the following form:

```
repo COMMAND OPTIONS
```

Optional elements are shown in brackets [ ]. Once Repo is installed, you can get information about any command by running

```
repo help COMMAND
```

Many commands take a project list as an argument. You can specify project-list as a list of names or a list of paths to local source directories for the projects:

```
repo sync [PROJECT0 PROJECT1 ... PROJECTN]
repo sync [/PATH/TO/PROJECT0 .../PATH/TO/PROJECTN]
```

init

```
$ repo init -u URL [OPTIONS]
```

Installs Repo in the current directory. This creates a `.repo/` directory that contains Git repositories for the Repo source code and the standard Android manifest files. The `.repo/` directory also contains `manifest.xml`, which is a symlink to the selected manifest in the `.repo/manifests/` directory.

Options:

- `-u`: specify a URL from which to retrieve a manifest repository. The common manifest can be found at `https://android.googlesource.com/platform/manifest`
- `-m`: select a manifest file within the repository. If no manifest name is selected, the default is default.xml.
- `-b`: specify a revision, i.e., a particular manifest-branch.

Note: For all remaining Repo commands, the current working directory must either be the parent directory of `.repo/` or a subdirectory of the parent directory.

sync

`repo sync [PROJECT_LIST]`

Downloads new changes and updates the working files in your local environment. If you run `repo sync` without any arguments, it will synchronize the files for all the projects.

When you run `repo sync`, this is what happens:

- If the project has never been synchronized, then `repo sync` is equivalent to `git clone`. All branches in the remote repository are copied to the local project directory.
- If the project has already been synchronized once, then `repo sync` is equivalent to:

  ```
  git remote update
  git rebase origin/BRANCH
  ```

  where `BRANCH` is the currently checked-out branch in the local project directory. If the local branch is not tracking a branch in the remote repository, then no synchronization will occur for the project.
- If the git rebase operation results in merge conflicts, you will need to use the normal Git commands (for example, `git rebase --continue`) to resolve the conflicts.

After a successful `repo sync`, the code in specified projects will be up to date with the code in the remote repository.

Options:

- `-d`: switch specified projects back to the manifest revision. Helpful if the project is currently on a topic branch, but the manifest revision is temporarily needed.
- `-s`: sync to a known good build as specified by the manifest-server element in the current manifest.
- `-f`: proceed with syncing other projects even if a project fails to sync.

upload

`repo upload [PROJECT_LIST]`

For the specified projects, Repo compares the local branches to the remote branches updated during the last repo sync. Repo will prompt you to select one or more of the branches that have not yet been uploaded for review.

After you select one or more branches, all commits on the selected branches are transmitted to Gerrit over an HTTPS connection. You will need to configure an HTTPS password to enable upload authorization. Visit the Password Generator to generate a new username/password pair to use over HTTPS.

When Gerrit receives the object data over its server, it will turn each commit into a change so that reviewers can comment on each commit individually. To combine several "checkpoint" commits together into a single commit, use git rebase -i before you run repo upload.

If you run repo upload without any arguments, it will search all the projects for changes to upload.

To make edits to changes after they have been uploaded, you should use a tool like `git rebase -i` or `git commit --amend` to update your local commits. After your edits are complete:

- Make sure the updated branch is the currently checked out branch.
- Use `repo upload --replace PROJECT` to open the change matching editor.
- For each commit in the series, enter the Gerrit change ID inside the brackets:

```
# Replacing from branch foo
[3021]35f2596cRefactor part of GetUploadableBranches
to lookup one specific...
```

```
[2829] ec18b4ba Update proto client to support patch
set replacments
[3022] c99883fe Teach'repo upload --replace' how to
add replacement patch se...
# Insert change numbers in the brackets to add a new
patch set.
# To create a new change record, leave the brackets
empty.
```

After the upload is complete the changes will have an additional Patch Set.

diff

`repo diff [PROJECT_LIST]`

Shows outstanding changes between commit and working tree using `git diff`.

download

`repo download TARGET CHANGE`

Downloads the specified change from the review system and makes it available in your project's local working directory.

For example, to download change 23823 into your platform/frameworks/base directory:

```
$ repo download platform/build 23823
```

A `repo sync` should effectively remove any commits retrieved via `repo download`. Or, you can check out the remote branch; e.g., `git checkout m/master`.

*Note: There is a slight mirroring lag between when a change is visible on the web in Gerrit and when `repo download` will be able to find it for all users, because of replication delays to all servers worldwide.

forall

`repo forall [PROJECT_LIST]-c COMMAND`

Executes the given shell command in each project. The following additional environment variables are made available by `repo forall`:

- `REPO_PROJECT` is set to the unique name of the project.

- `REPO_PATH` is the path relative to the root of the client.
- `REPO_REMOTE` is the name of the remote sstem from the manifest.
- `REPO_LREV` is the name of the revision from the manifest, translated to a local tracking branch. Used if you need to pass the manifest revision to a locally executed git command.
- `REPO_RREV` is the name of the revision from the manifest, exactly as written in the manifest.

Options:

- `-c`: command and arguments to execute. The command is evaluated through `/bin/sh` and any arguments after it are passed through as shell positional parameters.
- `-p`: show project headers before output of the specified command. This is achieved by binding pipes to the command's stdin, stdout, and sterr streams, and piping all output into a continuous stream that is displayed in a single pager session.
- `-v`: show messages the command writes to stderr.

prune

```
repo prune [PROJECT_LIST]
```

Prunes (deletes) topics that are already merged.

start

```
repo start BRANCH_NAME [PROJECT_LIST]
```

Begins a new branch for development, starting from the revision specified in the manifest.

The `BRANCH_NAME` argument should provide a short description of the change you are trying to make to the projects.If you don't know, consider using the name default.

The `PROJECT_LIST` specifies which projects will participate in this topic branch.

Note: "." is a useful shorthand for the project in the current working directory.

status

`repo status [PROJECT_LIST]`

Compares the working tree to the staging area (index) and the most recent commit on this branch (HEAD) in each project specified. Displays a summary line for each file where there is a difference between these three states.

To see the status for only the current branch, run `repo status`. The status information will be listed by project. For each file in the project, a two-letter code is used:

In the first column, an uppercase letter indicates how the staging area differs from the last committed state.

letter	meaning	description
-	no change	same in HEAD and index
A	added	not in HEAD, in index
M	modified	in HEAD, modified in index
D	deleted	in HEAD, not in index
R	renamed	not in HEAD, path changed in index
C	copied	not in HEAD, copied from another in index
T	mode changed	same content in HEAD and index, mode changed
U	unmerged	conflict between HEAD and index; resolution required

In the second column, a lowercase letter indicates how the working directory differs from the index.

letter	meaning	description
-	new/unknown	not in index, in work tree
m	modified	in index, in work tree, modified
d	deleted	in index, not in work tree