Git

1. Introduction

Definition 1.1 Revision: Represents a version of the source code. Git implements revisions as **commit objects** (or short commits). These are identified by an SHA-1 hash.

Definition 1.2 Commit: Commit holds the current state of the repository. A commit is also named by SHA-1 hash. When you commit your changes into a repository this creates a new *commit object* in the Git repository. This commit object uniquely identifies a new revision of the content of the repository. Every commit object has a pointer to the parent commit object. From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit. If a commit has multiple parent commits, then that particular commit has been created by merging two branches.

Definition 1.3 Branch: A branch is a named pointer to a commit. Selecting a branch in Git terminology is called to **checkout** a branch. If you are working in a certain branch, the creation of a new commit advances this pointer to the newly created commit.

Definition 1.4 Master: Is the default main branch that automatically gets created.

Definition 1.5 Staging Area/Index: The staging area is the place to store changes in the working tree before the commit. The staging area contains a snapshot of the changes in the working tree (changed, new or deleted files) relevant to create the next commit.

Definition 1.6 HEAD: Is a pointer, which (usually) always points to the latest commit *in a branch*. Whenever you make a commit, HEAD is updated with the latest commit. Think of the HEAD as the "*current checked-out branch*".
When you switch branches with `git checkout`, the HEAD revision changes to point to the tip/latest commit of the new branch.

How to see where HEAD points to?

```
cat .git/HEAD
```

Definition 1.7 Detached HEAD mode: Normally, when checking out a proper branch name, Git automatically moves the HEAD pointer along when you create a new commit and sets it to the lastest commit.
But: if we checkout a specific commit by a *SHA1 hash*, Git will not do this for us and thus if we want to commit new changes the no longer belong to any branch ⇒ detacht HEAD. In order to fix this we simply need to checkout a branch: e.g.

```
git checkout master
```

Notes

- There can exist multiple heads but the alias HEAD is the currently selected head.
  Hence: HEAD determines which branch we are currently on.

2. Git Config

Allows to get and set repository or global options.

Listening 1.1

```
dfsdf
```

Creation

0.1. Creating a gitlab project from the commandline

Creating remote Project

```
git push --set-upstream remote:namespace/project_name.git
↪  branch
```

- remote: e.g. git@gitlab.ethz.ch
- namespace: e.g. groups and subgroups
- branch: e.g. master

Note

Afterward simply add

```
git remote add origin remote/namespace/project_name.git
```

1. History

Log

Get a list of commit history:

```
magit-log [--all]
```

1.1. Undo local changes

Of local unstaged file

```
git checkout [--] filename
```

When checking out previous commit

```
git checkout -f master
```

1.2. Undo Commited Changeds

Undo without leaving a trace of the commit

1. Reset current HEAD to the specified state:
   ```
   git reset previous label or sha1
   ```
2. Do local work and commit it
3. Push and force remote to consider this push and remove the previous one (specifying remote-name and branch-name is not mandatory but is recommended to avoid updating all branches with update flag).
   ```
   git push -f remote-name branch-name
   ```

Note

Don't do this if someone allready pulled your change (I would use this only on my personal repo).

Local

0.1. Editing

Updating locals remote repository location

```
git remote set-url origin new_url
```

Unstaging a File

Unstage single file

```
git checkout file
```

Unstaging all Files

Unstage all files

```
git checkout --
```

Reset local unstaged changes to the last commit

```
git reset --hard HEAD
```

or equivalently

```
git checkout [-f/--force] HEAD
```

Note

While git checkout does only delete local changes, git reset will discard also commits if we go back in the history.

Overwrite local unstaged files with remote

1. Download content from remote without merge or rebase:
   ```
   git fetch --all
   ```
2. Reset the branch branch to what we just fetched from the remote:
   ```
   git reset --hard origin/branch
   ```

Attention hard

Will overwrite all local unstated and uncommitted changes.

Remote

List all remote's

```
git remote options
```

Options:

- --verbose/-v: show fetch&pull url's

Branches

List branches

```
git branch branch
```

Options:

- --all/-a: list both remote-tracking and local branches

Switching branches

```
git checkout branch
```

Options:

- -b: create a new branch if it does not exist

Tracking remote branches

```
git checkout -b branch --track remote/branch options
```

this is equivalent to to

```
git checkout branch
```

if branch does not exist locally but there exists exactly one remote (e.g. origin) with a matching name.

Options:

- --track/-t: specifies which remote branch to track if there exist multiple remotes with the desired branch. (implies -b for convenience).

Merging branches

Merging

Merging branches

1. checkout branch into which we want to merge the changes e.g. master:
   ```
   git checkout branch
   ```
2. Merge another branch into branch
   ```
   git merge another_branch
   ```

Submodules

1. Adding a new submodule

Run from main project repository:

```
git submodule add <path-to-external> [name]
```

2. Pushing new changes to the Submodule

Add, commit, pull, push normally from within the submodule.

```
1: cd [path-to-submodule]
2: git add [files]
3: git commit -m <message>
4: git pull [remote e.g origin] [branch e.g master]
5: git push [remote e.g origin] [branch e.g master]
```

3. Telling the project that the submodule changed

After making changes to the submodule normally add, commit, push submodule form main-project.

```
1: cd ..
2: git add <submodule>
3: git commit -m ``Updated reference to <name> submodule''
4: git pull [remote e.g origin] [branch e.g master]
5: git push [remote e.g origin] [branch e.g master]
```

4. Updating And Existing Submodule

To Update a project to the newest commit of the submodule run: `git submodule update [--init]` sometimes it is necessary to run
`git submodule update [--init --force --remote]`

Note

All git commands executed from within a submodule are with respect to that submodule repo and are not seen/affacted by the main repository.

Secure Shell (SSH)

1. Creating SSH Keys

Creating SSH Keys — Listening 8.1

```
cd ~/.ssh
# Create new key (will promt for filename)
ssh-keygen -t rsa -C "your_email@example.com"
# Print key
cat ~/.ssh/id_rsa.pub
# Copy key to remote server into .ssh/authorized_keys
ssh-copy-id -i ~/.ssh/id_rsa_key_name.pub username@host
```

2. Creating a SSH config file

Allows you to simply type `ssh a_name` (without any password prompt).

Creating SSH Keys — Listening 8.2

```
cd ~/.ssh
vim config
# enter the following into the file
Host a_name
     HostName hostname
     User username
     IdentityFile .ssh/id_rsa_key_name
```

hostname: Specifies the real hostname. Either as name or IP address.

Magit

Maggit Status C-x g

- **Magit Status**:

Useful Commands

- `magit-remote-add`: add new remote to git repository.
- `magit-log`: show commit log.

1. Show Remote Branches

`magit-show-refs-popup` or y during maggit status

2. Pull from Upstream

F: to display pull popup (`magit-pull-popup`)

3. Push to Upstream

1. **Staging**:
   s: to add the file under the cursor to the stage (`magit-stage`)
   S: to add all tracked files to the stage (`magit-stage-modified`)
   u: to unstage the file under the cursor to the stage (`magit-unstage`)
   U: to unstage all staged files (`magit-unstage-all`)
2. **Committing**:
   c c: write a commit message
   C-c C-c: Finish/close message and commit (`magit-commit-popup`)
3. **Pushing to upstream**:
   P u: push upstream (`magit-push-popup`)
4. **Pulling from upstream**:
   F: to pull (`magit-pull-popup`)

3.1. Submodules

`magit-submodule-popup`: show submodule status:

- a: add a submodule to the repository (`magit-submodule-add`)