Brief introduction to Git

Oana Cocarascu

Git is a Version Control System (VCS) that allows you to manage changes and different versions of files. VCS also simplifies working in group projects as it allows multiple people to work on the same files at the same time. In programming, it is often the case that you want to save a working state of the code. VCS is useful as it maintains a history of the changes and it allows developers to revert and to go back to an older version of the code. You can check a snapshot of your code at any time and see the difference between different snapshots (called "versions" or "commits").

A Git repository (repo) consists of the following main components:

- *working tree* (or *working directory*) consists of the files that you are currently working on. It also contains a `.git` directory with the index and object store
- *index* (or staging area) for the files you wish to record in the object store
- *object store* that keeps track of all snapshots

The basic Git workflow is as follows: you modify files in the working tree, you stage the changes you want to be included in the next commit, and you commit these changes with a message describing what you changed. Committing means taking the files from the index and storing them as a snapshot in the repository. The object store records snapshots of the files after each commit and each snapshot knows the snapshots before and after it. You can checkout a recorded snapshot from the object store. This will update the working directory to reflect that version of the project.

To run the following commands you need to be inside a working directory of a Git repository:

- `git help` explains what commands do. You can prefix any command by help to get help specific to that command (e.g. `git help add`).
- `git init <directory>` creates a new directory as a working directory and initialises an index and an empty object store.
- `git clone <repository> <directory>` creates a clone of repository under the given directory. If the directory is omitted, Git creates the clone under the same name as the name of the repository.
- `git status` shows the status of the project. You can see which files in the working directory have changes not recorded in the index, which branch you are working on, and which files are not tracked by Git.
- `git diff` shows what has been created or changed.
- `git log` shows the snapshots in the object store.
- `git add <filenames>` adds the given files to the index.
- `git commit -m "Message"` creates a new snapshot in the object store based on the files that have been added to the index. The message should be meaningful and should describe what changed as it will be associated with the snapshot. If `-m "Message"` is omitted, Git will start up an editor for you to type your message in.

- `git push` pushes snapshots from your local object store to an object store of a remote repository. If the repository was created using `git clone` then `git push` will push back to that.
- `git checkout` will undo changes you have made to files in your working directory, reverting the files back to the version in `HEAD` (`HEAD` is a reference to the last commit).
- `git checkout <commitID>` will make the working directory reflect the snapshot at `commitID` which can be found using `git log`.

Until now you have been working on the `master` branch. If you want to work on a feature, you can create a branch and work on that branch. Once your code is ready, you can merge that branch into `master`.

Figure 1 shows the workflow in basic branching. At `Common base`, a new branch is created. Two more commits are pushed to the `master` branch (using `git add`, followed by `git commit` and `git push`). Three commits are pushed to the other branch (similarly using `git add`, followed by `git commit` and `git push`).

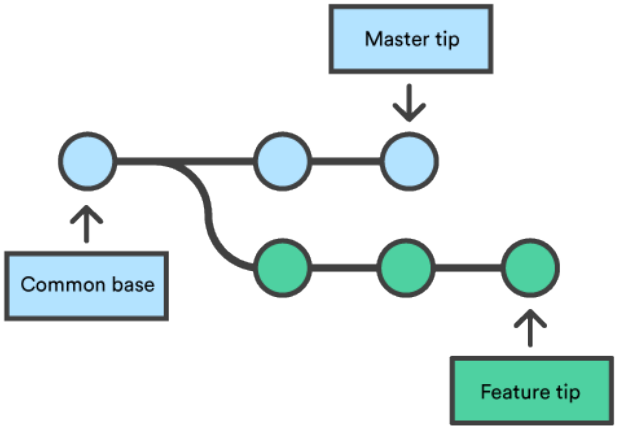

Figure 1: Basic branching

- `git checkout -b feature` creates a new branch and switches to it at the same time (note that I named the branch `feature`, you can give any name you like)
- `git branch feature` creates a new branch `feature` but does not switch to that branch
- `git checkout feature` switches to the `feature` branch. Thus if you are on a different branch and want to switch to `master`, you will run `git checkout master`

Figure 2 shows what happens when you merge into the `master` branch.

Make sure you do not have any uncommited changes before merging (there are solutions for this but we are looking at basic branching/merging). Assuming you are on the `feature` branch, to merge that branch into `master` you need to run:

- `git checkout master` that switches to the `master` branch. Note that git resets your working directory to look like it did the last time you committed on that branch
- `git merge feature` that merges the `feature` branch into the current branch (`master` in this case as we already switched to the `master` branch)

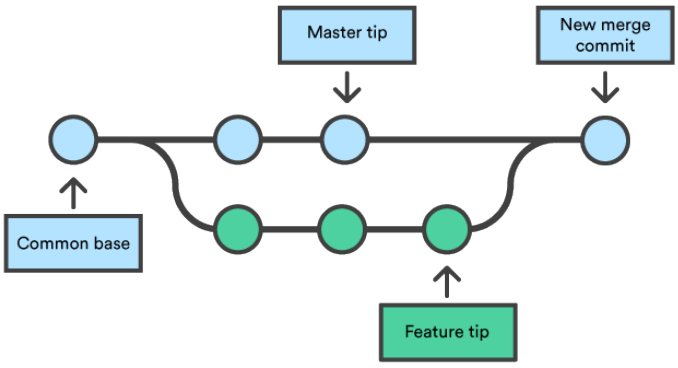

Figure 2: Basic merging

Sometimes the merging does not go smoothly. This happens if you changed the same part of the same file differently in the two branches you are trying to merge. You will see something similar to:

```
Auto-merging some_file.py
CONFLICT (content):  Merge conflict in some_file.py
Automatic merge failed; fix conflicts and then commit the result.
```

If you type `git status`, you will see:

```
both modified:  some_file.py
```

Git adds standard conflict-resolution markers to the files that have conflicts so it makes it easier to resolve conflicts. Your file will have a section similar to:

```
<<<<<<< HEAD:some_file.py
here you have the version on master branch because master is the branch where you ran the merge
command
=======
here you have the version from the feature branch, the one you are trying to merge
>>>>>>> feature:some_file.py
```

To fix this, keep only the code in the version you want to be on the `master` branch, remove the markers, and then run `git add` on the file. Do this for every file that could not be merged.

For more information about Git commands you can:

- run `man gittutorial` in a terminal
- check Git page `https://git-scm.com/`