Lecture 4: Branching
Last time

- How Git handles commits
- Git commit history
- Undoing changes
- Rewriting history
Today

At the end of today’s lesson, you will know:

- What are branches in Git
- Why and how to use branches
- Commands for branch management
- Common branching models
Situation until now

- Until now the commit history has been linear

- The main flow of history is the main branch, often called “master branch”
- This name can be changed, but is usually left at it to avoid confusion
Branches in a nutshell

- A “branch” is where history diverges from the master branch
Why use branches

- Branches are used to separate different tasks
- Separate your work from the work of others
- When working on fixes or new features, do that in a separate branch
- Good for experiments, big reworks etc
- Generally the master branch should contain the latest stable version
- When a fix/feature is ready and tested, merge into master
Git branching: current situation

HEAD is a pointer to the latest committed state - essentially “where we are right now”
Showing a list of branches

```
$ git branch
* master
```
Adding new branch

$ git branch newbranch
$ git branch
  * master
    newbranch
Switching branches

$ git checkout newbranch
Switched to branch 'newbranch'

$ git branch
  master
  * newbranch
Branches at the core of Git mantra

- In earlier VCS, branching was harder and had a big overhead
- Creating a branch meant copying the whole project into another folder
  - Seconds or even minutes
- In Git, branches are just pointers to certain commits
- → Virtually no overhead in computation and memory
- Created, deleted locally
- → Near-instantaneous

- Git mantra: “branch early; branch often”
Committing on a branch

$ git commit -m "Commit 5"
[newbranch 5dd101030] Commit 5
$ git branch
  master
  * newbranch
Back to master

$ git checkout master
Switched to branch 'master'
History diverges

$ git commit -m "Commit 6"
[master d433570] Commit 6
Pushing branches to remote repository

- In Git, all changes are made locally, unless you explicitly push them
  - Adding and committing files
  - Creating branches
- New branches will not be automatically pushed
- You can have local branches you use for testing/experiments
- When ready, need to be explicitly pushed to share:

$ git push <remote> <branch>
Viewing history

$ git branch
* master
  newbranch
$ git log --oneline
  d433570 Commit 6
  0a52e95 Commit 4
  621c107 Commit 3
  6c79b23 Commit 2
  26b45c8 Commit 1
History of all branches

$ git log --oneline --all

d433570  Commit 6
58d06a1  Commit 5
0a52e95  Commit 4
621c107  Commit 3
6c79b23  Commit 2
26b45c8  Commit 1
Visualise branches

```bash
$ git log --oneline --all --graph
* d433570 Commit 6
  * 58d06a1 Commit 5
  /
* 0a52e95 Commit 4
* 621c107 Commit 3
* 6c79b23 Commit 2
* 26b45c8 Commit 1
```
Visualise branches

$ git log --oneline --all --graph --decorate
* d433570 (HEAD -> master) Commit 6
  | * 58d06a1 (newbranch) Commit 5
  |
  * 0a52e95 Commit 4
  * 621c107 Commit 3
  * 6c79b23 Commit 2
  * 26b45c8 Commit 1
Power of Git aliases

$ git config --global alias.history "log --oneline --all --graph --decorate"

$ git history
* d433570 (HEAD -> master) Commit 6
  * 58d06a1 (newbranch) Commit 5
/ *
* 0a52e95 Commit 4
* 621c107 Commit 3
* 6c79b23 Commit 2
* 26b45c8 Commit 1
Back to branches
Merging

- Branches can be “merged” back into the master branch

- This is how a commit can have more than one parent commit
Merging branches

$ git merge newbranch
Merge made by the ‘recursive’ strategy.

- “Merge given branch into current branch”
$ git history

* 3d859db (HEAD -> master) Merge branch 'newbranch'
  |
  * 58d06a1 (newbranch) Commit 5
  * d433570 Commit 6
  |
  * 0a52e95 Commit 4
  * 621c107 Commit 3
  * 6c79b23 Commit 2
  * 26b45c8 Commit 1
Branches can come out of other branches
Complex Merging

- Of course they can be merged into each other
Complex Merging

- Branches don’t have to be abandoned after being merged once
$ git branch -d newbranch
Deleted branch 'newbranch' (was 58d06a1)
$ git branch
* master
History after branch deletion

$ git history
* 3d859db (HEAD -> master) Merge branch 'newbranch'
|\  
| * 58d06a1 Commit 5
* | d433570 Commit 6
|/
* 0a52e95 Commit 4
* 621c107 Commit 3
* 6c79b23 Commit 2
* 26b45c8 Commit 1
Deleting branches in remote

- As with adding, deleting only deletes the local branch
- History is preserved
- Will warn when deleting branch that contains unmerged changes:

```bash
$ git branch -d newbranch
error: The branch 'newbranch' is not fully merged.
If you are sure you want to delete it, run 'git branch -D testing'.
```

- Should not be feared
- Deleting remote branch:

```bash
$ git push --delete <remote> <branch>
```
Recap

- We can **create** branches (e.g. branch out from master)
  - We can branch out recursively (branch out from a branch)
- We can **switch** between branches
- We can **view** the history of branches
- We can **merge** two branches (with a common ancestor)
- We can **delete** branches
- And now, what do we do with all this?
  - When should I branch?
  - When should I merge?
Branching models

- Smaller/solo projects:
  - Doing most of work in master
  - Creating branches for bigger tasks
  - Naming branches as seems appropriate

- Companies or bigger open-source projects usually have special guidelines

- Policy may enforce:
  - How to name branches
  - How often to create branches
  - How to merge branches into master
Common guidelines

- Master branch contains the latest “release” version of the product
- Never make commits to master directly
- To add commits, make a new branch, make commits and merge it
- Sometimes only privileged people are allowed to merge into master
- Others need to make “merge requests” when their branches are ready
Types of branches

- Develop branch
- Topic branch
- Hotfix branch
- Release branch
- Depending on the project, may be others
Develop branch

- Usually called “develop”, “integration”, “dev” or something similar
- Branches out from master
- Merges into master when it’s time to release
- Contains the latest stable “development” version of the product
- Is the source of topic branches
- Is not deleted after it’s merged
- Typically don’t commit to develop either - make commits in topic branches
Topic branch

- Branches out from develop
- Merges into master when feature/bugfix is complete
- Contains commits for one particular requested feature or bugfix for the project
- Can be deleted once the changes are merged into development branch
- Commits are made directly into it
- Can spawn more topic branches and can be merged into parent topic branch
Topic branch naming conventions

- Usually prefixed with “feature” or something similar
- Has ID number of the issue it’s made to solve
- Typically includes short description

feature/23-fix-header-images
feature_123
fb-51_add-footer-contact-info
f9123-signup-form
Topic branches example
Hotfix branch

- Branches out from master
- Merges into master and develop
- Used when an urgent problem is discovered in released version of the product
- Deleted after merge
Release branch

- Branches out from develop
- Merges into develop and master when ready for release
- Contains the final feature-set for release
- Created for planning release and applying last moment fixes before releasing a new version
Branches in Github

Zürich design for Voog

- 304 commits
- 4 branches
- 16 releases
- 5 contributors

Branch: master

SSirjes: Added CDN for jquery (#106)

Latest commit 411d014 10 days ago
Branches in Github - switching branch
## Branches in Github - branch browser

### Default branch

<table>
<thead>
<tr>
<th>Branch</th>
<th>Updated</th>
<th>Author</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>master</td>
<td>10 days ago</td>
<td>SSirjes</td>
<td></td>
</tr>
</tbody>
</table>

### Active branches

<table>
<thead>
<tr>
<th>Branch</th>
<th>Updated</th>
<th>Author</th>
<th>PRs</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>108_Signout_Button</td>
<td>13 hours ago</td>
<td>srimha</td>
<td>0/4</td>
<td>Open</td>
</tr>
<tr>
<td>92_Search_noResults_translation</td>
<td>7 days ago</td>
<td>srimha</td>
<td>0/5</td>
<td>Open</td>
</tr>
</tbody>
</table>

### Stale branches

<table>
<thead>
<tr>
<th>Branch</th>
<th>Updated</th>
<th>Author</th>
<th>Merged</th>
<th>PRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>90_mixin_depr</td>
<td>3 months ago</td>
<td>Alvar Anthony H...</td>
<td></td>
<td>12/0</td>
</tr>
</tbody>
</table>


Real-world branching models

- [https://barro.github.io/2016/02/a-successful-git-branching-model-considered-harmful/](https://barro.github.io/2016/02/a-successful-git-branching-model-considered-harmful/)
Additional reading

https://learn-git-branching.js.org/

Interactive tutorial on Git branching, including more advanced topics like rebasing.
Practical session

- Listing; creating; deleting branches
- Switching branches
- Merging branches
References

- http://nvie.com/posts/a-successful-git-branching-model/