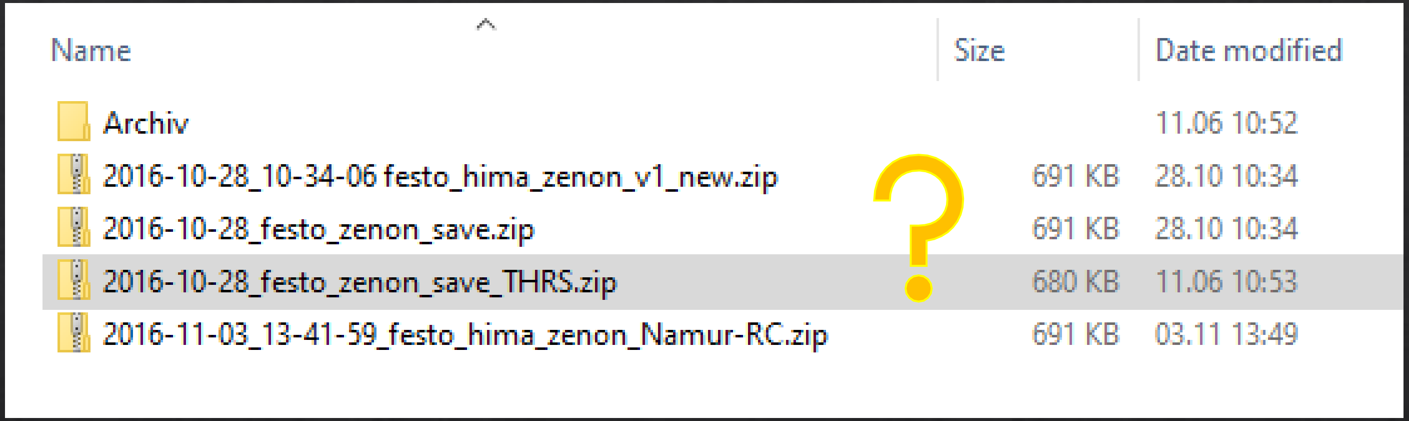
git

There's more than one way to skin a cat*

* No cats have been harmed in the creation of this material

Why do we want Version Control?



Name	Size	Date modified
Archiv		11.06 10:52
2016-10-28_10-34-06 festo_hima_zenon_v1_new.zip	691 KB	28.10 10:34
2016-10-28_festo_zenon_save.zip	691 KB	28.10 10:34
2016-10-28_festo_zenon_save_THRS.zip	680 KB	11.06 10:53
2016-11-03_13-41-59_festo_hima_zenon_Namur-RC.zip	691 KB	03.11 13:49

- Everybody benefits from a version control system (vcs)
- git is the most common vsc
- git is finally standard at Festo
- git is fun and there is always something new to learn!

What will we look at?

- Basics
  - Repositories
  - Commits
  - Branches
  - Rebase or merge
  - What to do if it goes wrong
  - Do’s, Don’ts

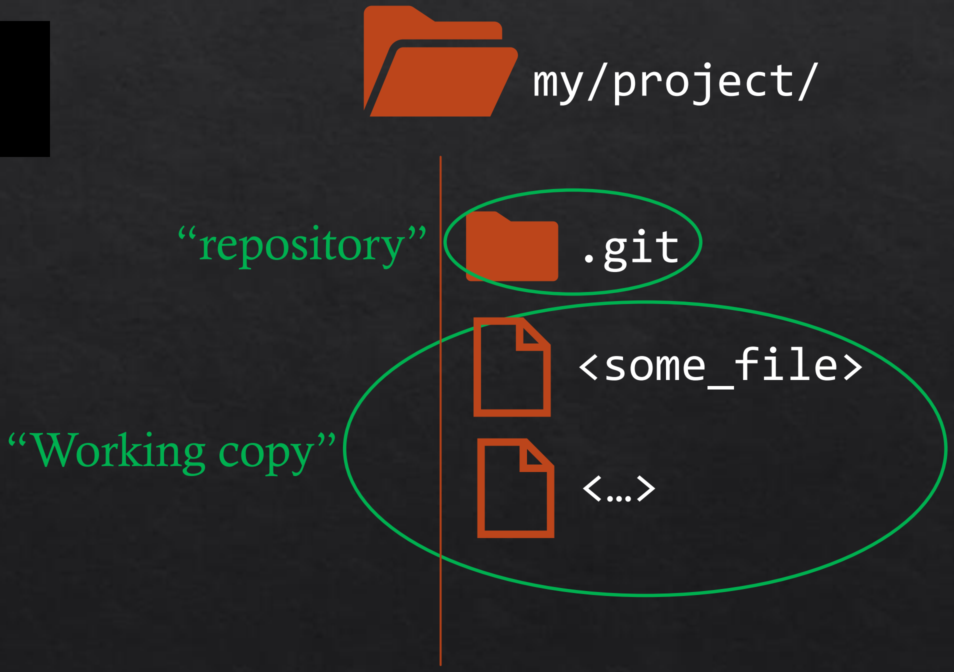
- Interactive live session – let’s play together

Basics – the repository

- Repository = Database of changes
- How to create a repository:

```
$ cd my/project/directory
$ git init
```

- Repository: git's files – **do not edit**!
- Working Copy: your files

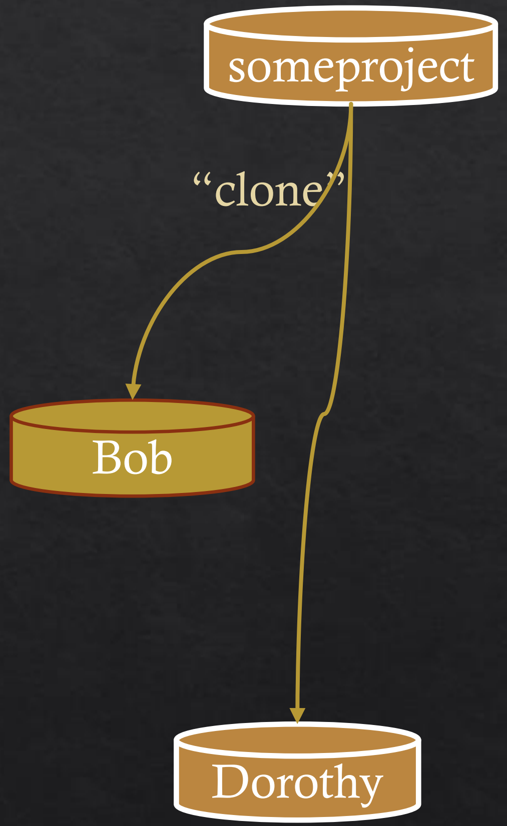


The repository – Cloning or “Forking”

- Clone an existing ‘remote’ repository
  - Creates an exact copy of the repository
  - Can evolve independently – “forking”

```
$ cd my/projects/
$ git clone https://adegit01.de.festo.net/someproject
$ cd someproject
$ git remote -v
origin  https://adegit01.de.festo.net/someproject(fetch)
origin  https://adegit01.de.festo.net/someproject(push)
```

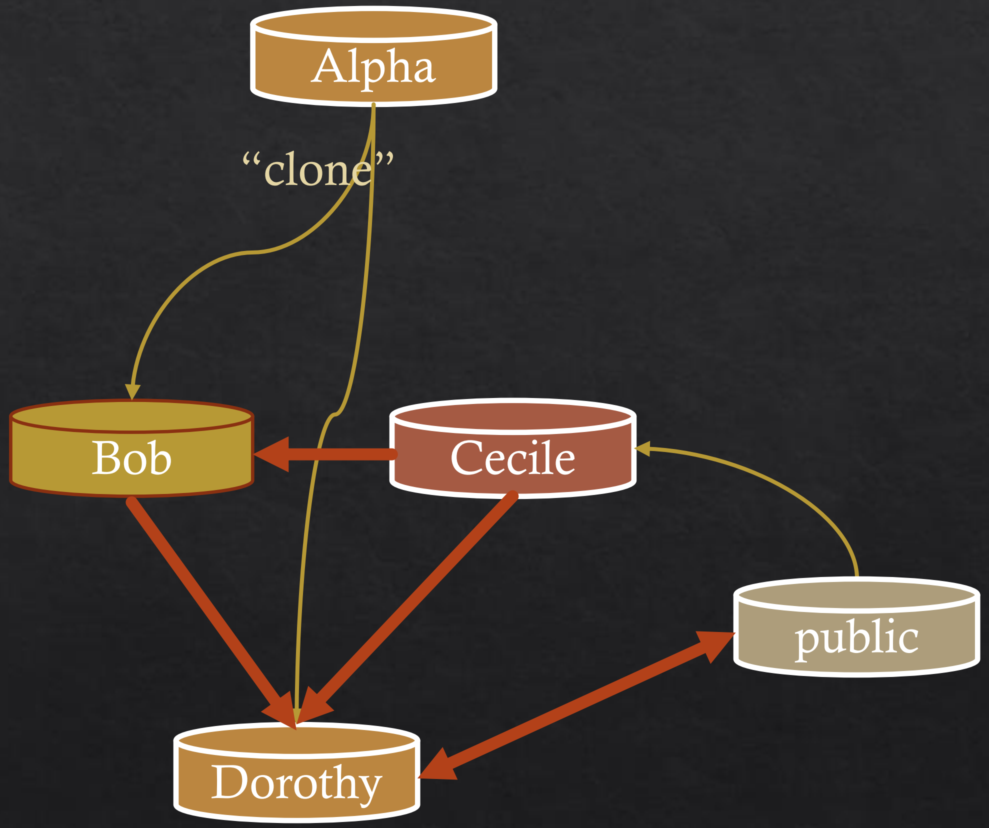
Name – “origin” by convention



→ The normal ‘corporate’ way

The repository – No single source of truth

- **Distributed** Version Control System
    - → all repos are equally good
- Alpha was "forked"
    - → Changes are not synced to origin



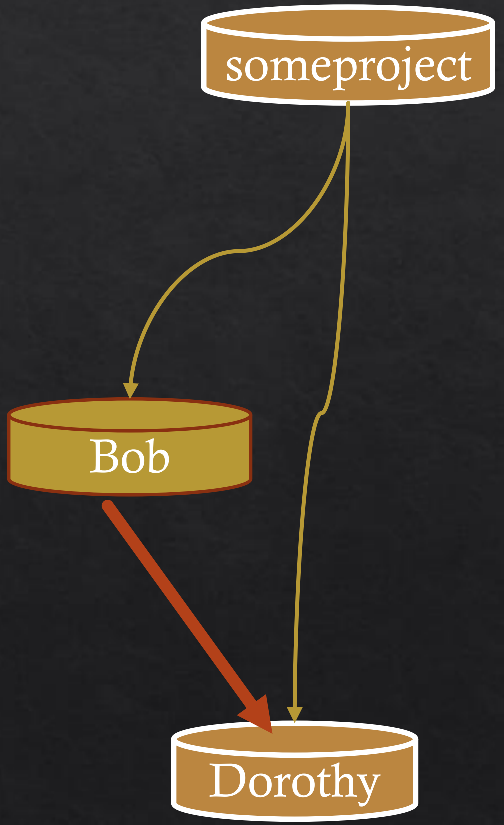
→ It all works because of a common history

The repository – connecting a repository

- Dorothy wants to get changes from Bob

```
$ git remote add bob https://bob.com/repos/someproject
$ git remote -v
origin  https://adegit01.de.festo.net/someproject(fetch)
origin  https://adegit01.de.festo.net/someproject(push)
bob     http://bob.com/repos/someproject (fetch)
bob     http://bob.com/repos/someproject (push)
```

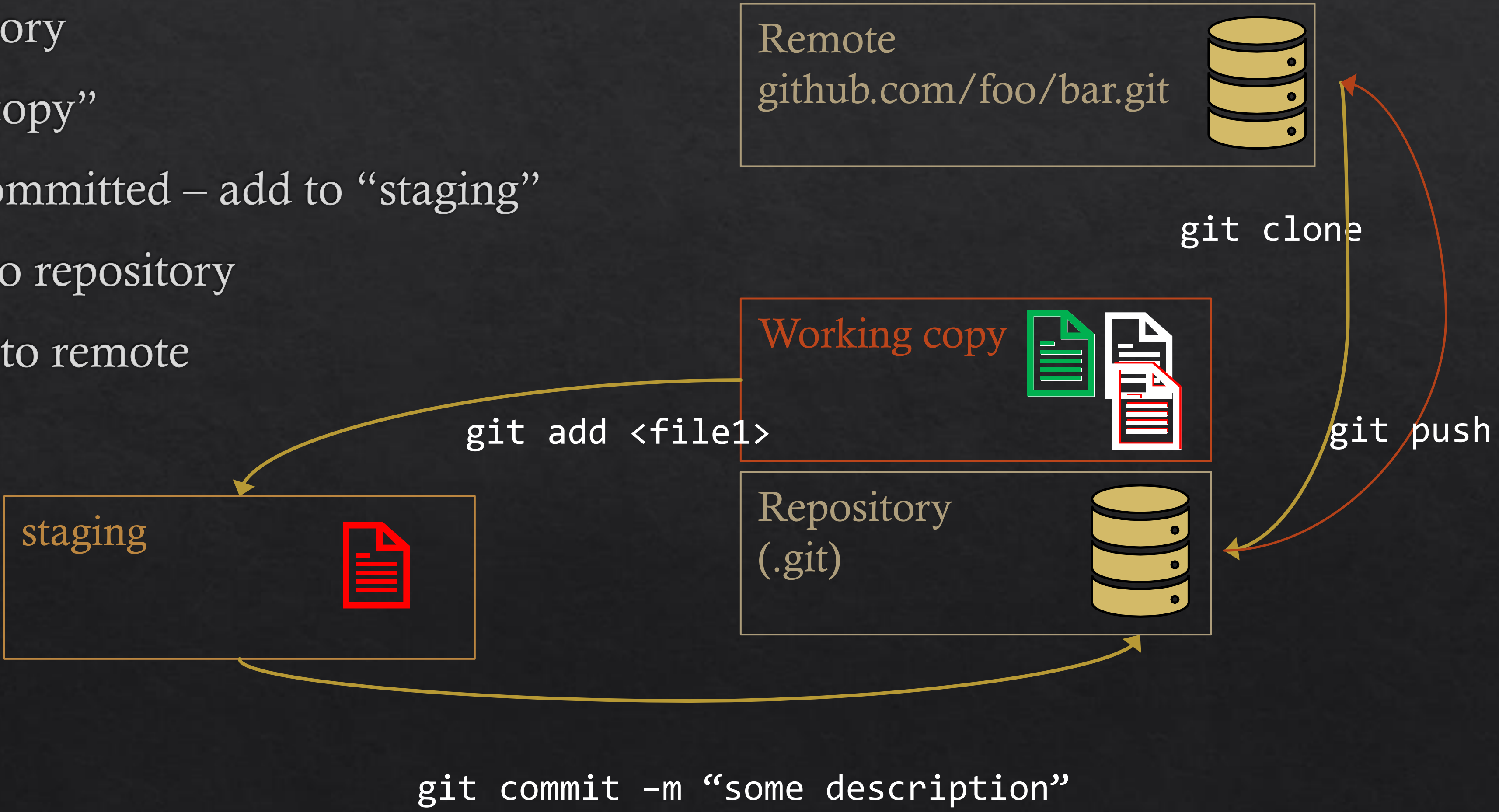
Just a name



→ When you join several repositories

The git life-cycle - overview

- Create or clone a repository
- Change files "working copy"
- Prepare changes to be committed – add to "staging"
- Commit changes – add to repository
- Publish commits – push to remote



Basics – what is the status

- Get information about the current state of your working copy and repository

```
$ git status
On branch foo
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Makefile

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   pois.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README
```

Basics – staging

- Saving work to repository is a two-step process
- Add things to staging

```
$ git add <path>
```

- Looks like adding a file but in fact only the **current diff** (untracked files are added fully)
- Add individual parts of a file – **you want a tool for this!**

```
$ git add -i
```

- Editing a file after `'git add'` will **not include new edits in commit**

→ Use staging to create clean, atomic commits

Basics – the commit*

- Something binding and fixed for eternity
- git commit saves **staged** changes to repository
- Needs a "commit message" = explain what has changed

```
$ git commit -m "some mAssaGe"
```

- Creates a unique "hash" – e.g. `9b7688fb`
- You can edit – "amend" last commit – break the promise!

```
$ git commit --amend -m "a better message"
```

4711bdf
9b7688fb some mAssaGe
a better message

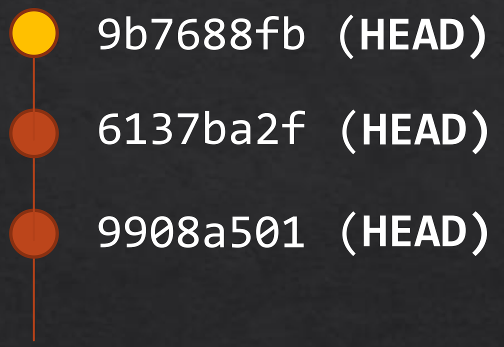
→ Commit often
→ Split larger changes in small coherent commits (use staging!)

*Commit [en] = festlegen, verpflichten [de]

The Commit-History

- A line of development –'DAG' (directed acyclic graph)
- Consists of commits identified by a hash
- HEAD pointer moves

- What can you do with a history?
  - Compare what has changed
  - You can time-travel!
  - Undo changes & Rewrite history



Commit History – What has changed

- Difference between last commit (HEAD) now (current working copy)

```
$ git diff
index 1200e8e..1f69a1f 100644
--- a/helm/values.yaml
+++ b/helm/values.yaml
@@ -11,7 +11,7 @@ image:
-  tag: "0.2.0"
+  tag: "0.2.0-20210615135255"
```

- What happened in a specific commit

```
$ git show 6137ba2f30f
```

- Difference between now and 2 arbitrary commits for a given file

```
$ git diff 6137ba2f30f 9908a5015ca0b -- Makefile
```

→ The diff shown is what happened from first to second reference

Commit History - Time travel

- Just check out a commit from the past
- Puts your working copy in the state of this commit (untracked files remain)

```
$ git checkout 9908a5015ca0b
Note: switching to '9908a5015ca0b2545b5eedbef72f980d8775ab33'.

You are in 'detached HEAD' state. You can look around, make
experimental changes and commit them, and you can discard any
commits you make in this state without impacting any branches
by switching back to a branch.
```

Don’t panic – it is git's way to tell you that you are not at a named 'branch’

→ Useful for exploring another solution

Commit History – Selective Time travel

- Just check out a file version of commit from the past

```
$ git checkout 6137ba2 -- Makefile
$ git status
On branch develop
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   Makefile
```

→ Recover deleted / changed files form the repository

Commit History – Undo and rewrite history

Do NOT do this in public! – Remember you committed to the changes

- Soft reset’
  - keep changes to the file(s) – as is on HEAD
  - Drop all intermediate commits between HEAD and reference (6137ba2)

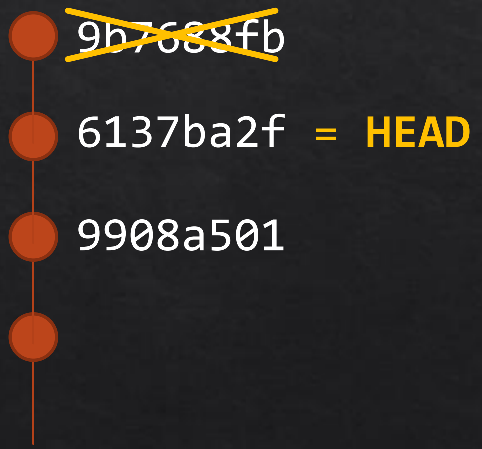
```
$ git reset 6137ba2
```

→ **Great for creating a clean commit history**

- Hard reset
  - Discard all changes and commits since reference (6137ba2)

```
$ git reset --hard 6137ba2
```

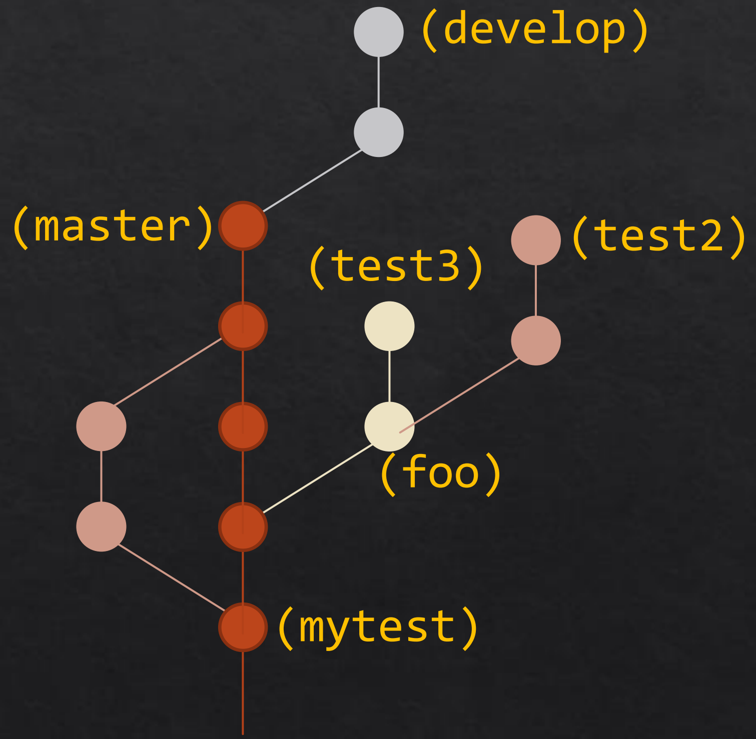
→ **Acts like checkout but erases history**



Branches

- Is a **named** for a line of commits – meaningful names!
- All branches have names - first branch is called “master”
- Name moves along with commits (name points to latest)
- “branch-point” can be any commit
- Can be deleted, renamed, merged and moved around

→ A great thing, elementary to git



Branches – Switching & Creating

- 'HEAD' moves – working copy content follows

```
$ git checkout develop
```

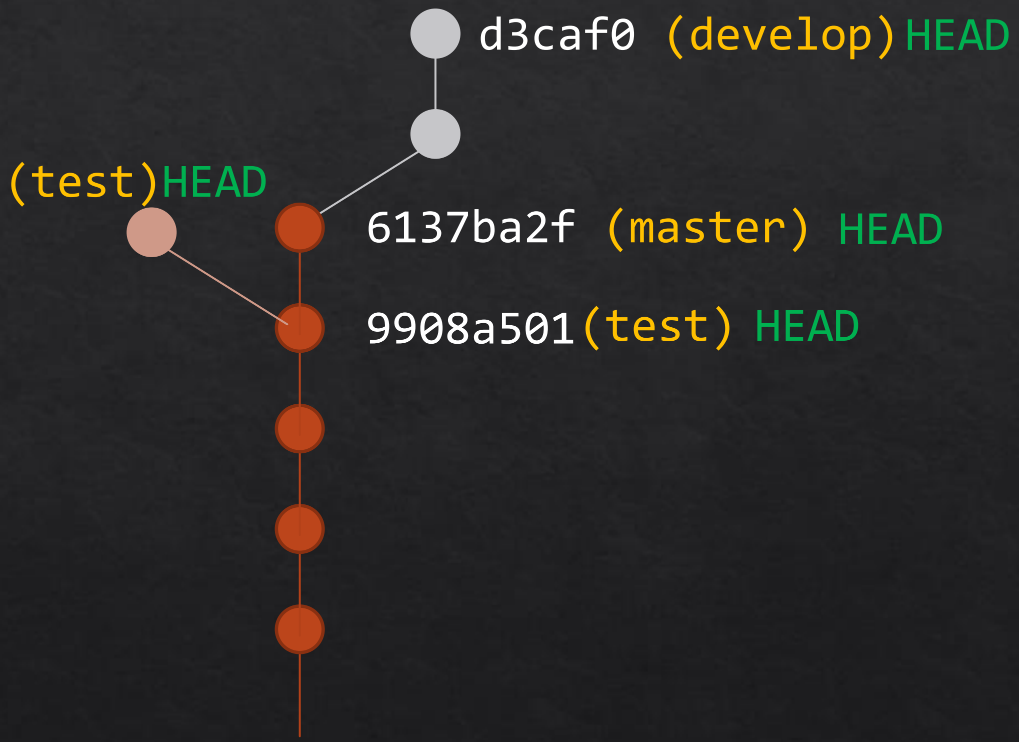
- Create a new branch on a commit

```
$ git checkout -b test 9908a501
```

- Create a new branch on a commit and move HEAD

```
$ git checkout -B test 9908a501
```

```
$ git commit …
```



Branches – Deleting

- Delete a local branch

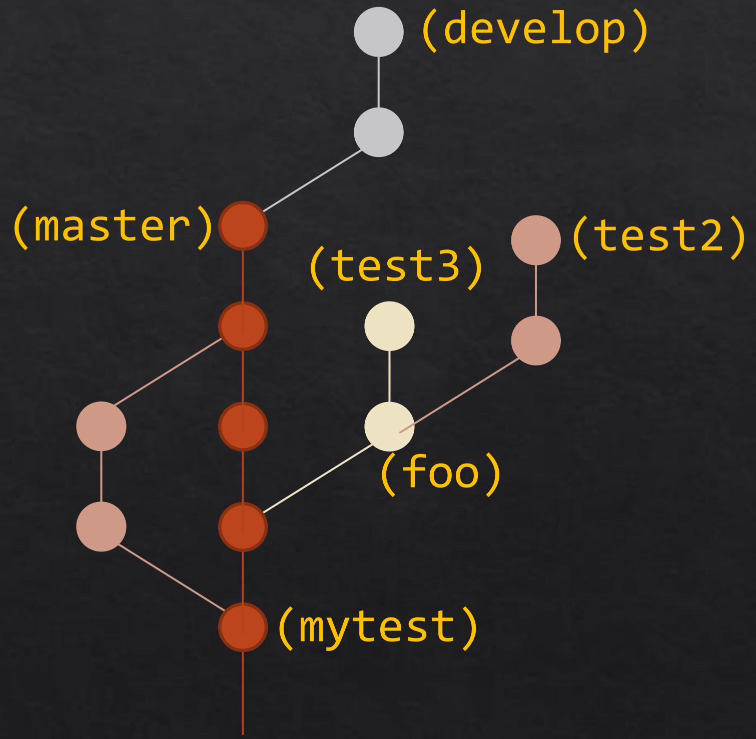
```
$ git branch -d test2
```

- Git will warn if the branch is not merged

```
error: The branch ‘test2' is not fully merged.
If you are sure you want to delete it, run
'git branch -D foo'.
```

- No danger in deleting ‘`mytest`’ – its just a label
- Delete a public/remote branch

```
$ git push -d origin test2
```



Branches – Merging

- Checkout the one to keep, Merge the other one

```
$ git checkout master
$ git merge develop
```

- Git resolves merges automatically (works pretty well)

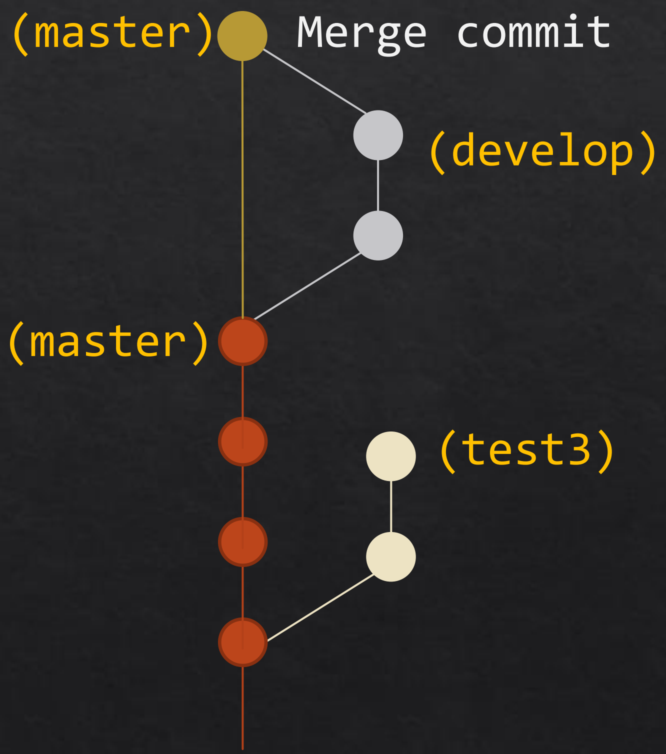
- If two branches edit the same lines you get a Merge Conflict

😟

```
$ git merge test3
```

- Check if you can merge

```
$ git merge --no-ff --no-commit test3
```



Merge Conflicts

- Use a manual merge-tool (kdiff3, meld, beyondcompare….)

```
$ git mergetool
```

- What if you can't merge?

```
$ git merge --abort
```

- What if you created a really bad merge?

```
$ git reset --hard <commit_before_merge>
```

- How to avoid merge conflicts for your colleagues?
- Spoiler: you will have to resolve the merge conflict:

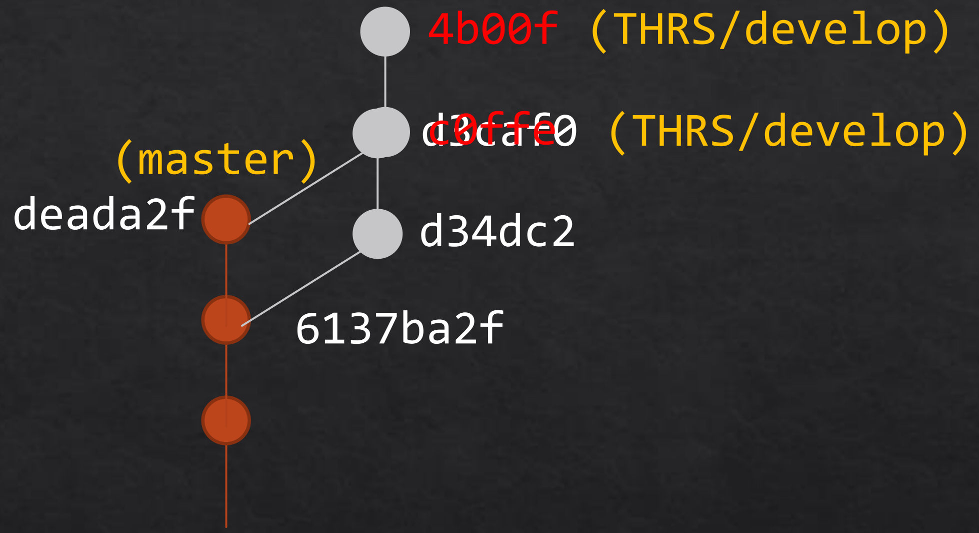
```
$ git checkout test3
$ git merge master (+ resolve conflicts)
$ git checkout master
$ git merge test3
```

Branches - Rebase

- What if we create a single line of commits?

  → Put your development on top

```
$ git checkout develop
$ git rebase origin/master
```



- Can go wrong – use a merge tool
- If it really goes wrong, start over:

```
$ git rebase --abort
```

Rewrites History – make sure others don't depend on it!

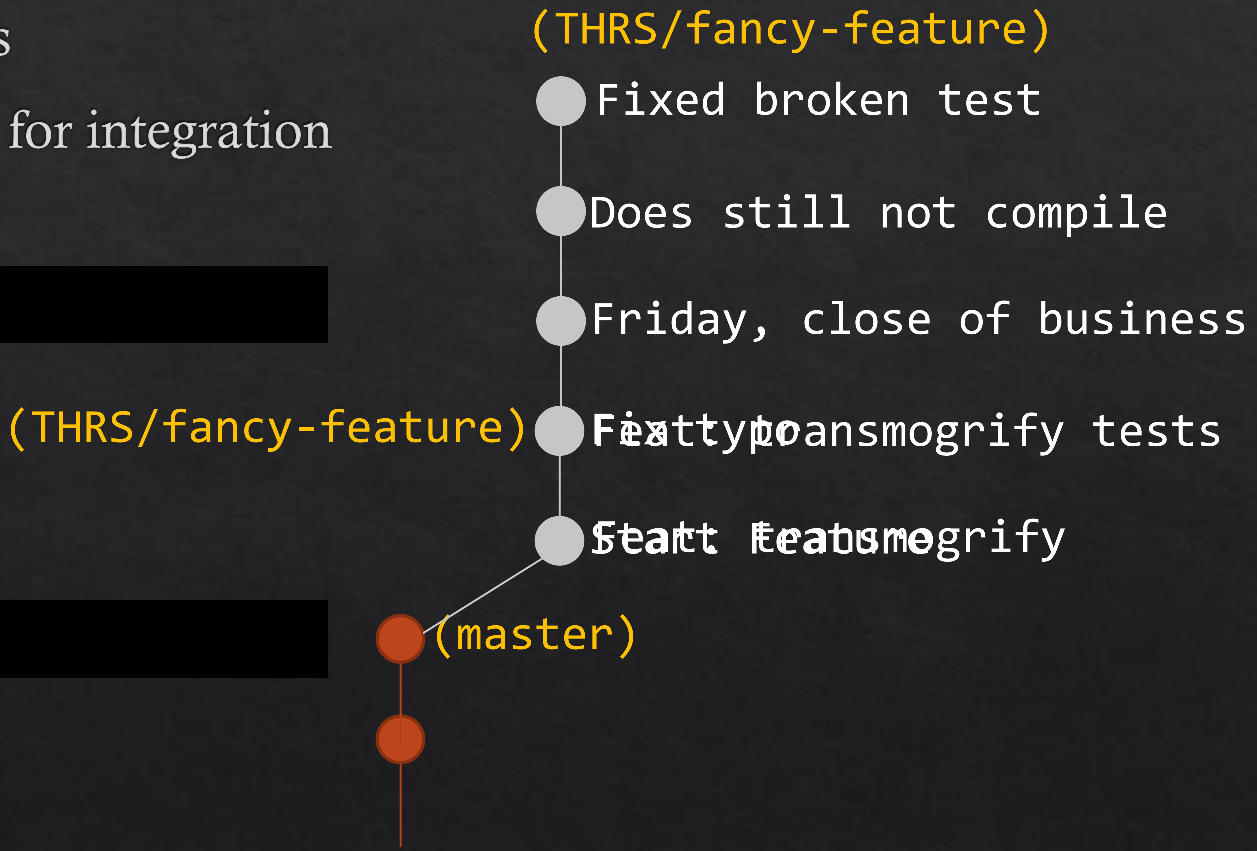
Rebase to clean up

- You have many work-in-progress commits
- You want one clean patch to make it easy for integration
- Choose and rephrase your commit

```
$ git rebase -i
```

Or the hard way – recreate all your commits

```
$ git reset --soft master
```



→ Please clean up your commits, it's easier to find problems later

Basics – remote branches

- Publish your changes to a remote ~~server~~ repository

```
$ git push origin HEAD
```

repository What?

- Update local repository (not working copy)

```
$ git fetch --all
```

- What branches exist on remote?

```
$ git branch -r
  bob/mytest
  bob/foo
  origin/HEAD -> origin/foo
  origin/develop
  origin/foo
  origin/master
```

Remote branches - Pull

- Getting changes from a tracking branch

```
$ git pull
```

- Will **merge** your working copy

```
$ git pull
There is no tracking information for the current branch.
Please specify which branch you want to merge with.
[…]
```

- Happens to me all the time… when you push your branch and pull again
- Set tracking information

```
git branch -u origin/mytest
```

OMG – I broke the repo!

- **Relax** – most likely you didn't break the repo, you only broke the working copy

```
$ git reset --hard <last_good_commit>
```

- I deleted a file some time ago – then check it out

```
$ git checkout <last_good_commit|branch> -- <filename>
```

- I deleted a branch

```
$ git reflog
$ git checkout <hash_from_reflog>
```

Do’s & Don’ts

- Do not publish all commits – create one clean patch (`rebase -i`)
- Do not change public commits other people use (includes rebase)
- Do not `--amend` on master after push!
- Do not base your work on temporary work of others
- Do not push to others’ branches

Do's & Don'ts

- Do Branch often, Commit more often
- Do read google and manuals
  - `git <command> -h`
  - https://git-scm.com/
  - https://github.com/k88hudson/git-flight-rules
  - https://learngitbranching.js.org/
  - https://lostechies.com/joshuaflanagan/2010/09/03/use-gitk-to-understand-git/
  - https://lostechies.com/joshuaflanagan/2010/09/03/use-gitk-to-understand-git-merge-and-rebase/
- Use a personal `.gitignore` file for files that your favorite tool creates
- Use a personal `.gitconfig` file with aliases for complicated commands
- **Experiment – don't be afraid**
- **Share with your colleagues – there is more than one way to skin a cat!**

That's all Folks!

….Not really, is it?

Well, what about….

- `git cherry-pick`
- `git am`
- `git --format-patch`
- `git email`
- and much more stuff I don’t even know exists….

- How to work with git?
  - **`git flow`**
  - forks and pull requests