Danske Bank

Introduction to Git

David Stiel
Danske Bank
dsti@danskebank.dk

Danske Bank

Prerequisites

- Nearly none
- Command line (powershell, bash or whatever)
- An editor

Goals

- Understand the concepts and terminology. So you can look for help your self
- Get comfortable with everyday git tasks
- Get into advanced stuff if time permits.

Exercise

Install Git

- Get on wifi
- Install git from https://git-scm.com/downloads/
- Start a terminal/command line write git.

About git clients

There are different git clients we will use the commandline one:

- It is you get help
- It is the only feature complete one
- It gives the best understanding
- It is where you can do the most advanced tricks
- It convenient for this audience

Often you use different clients for different tasks.

Knowledge

What is version control systems (VCS)

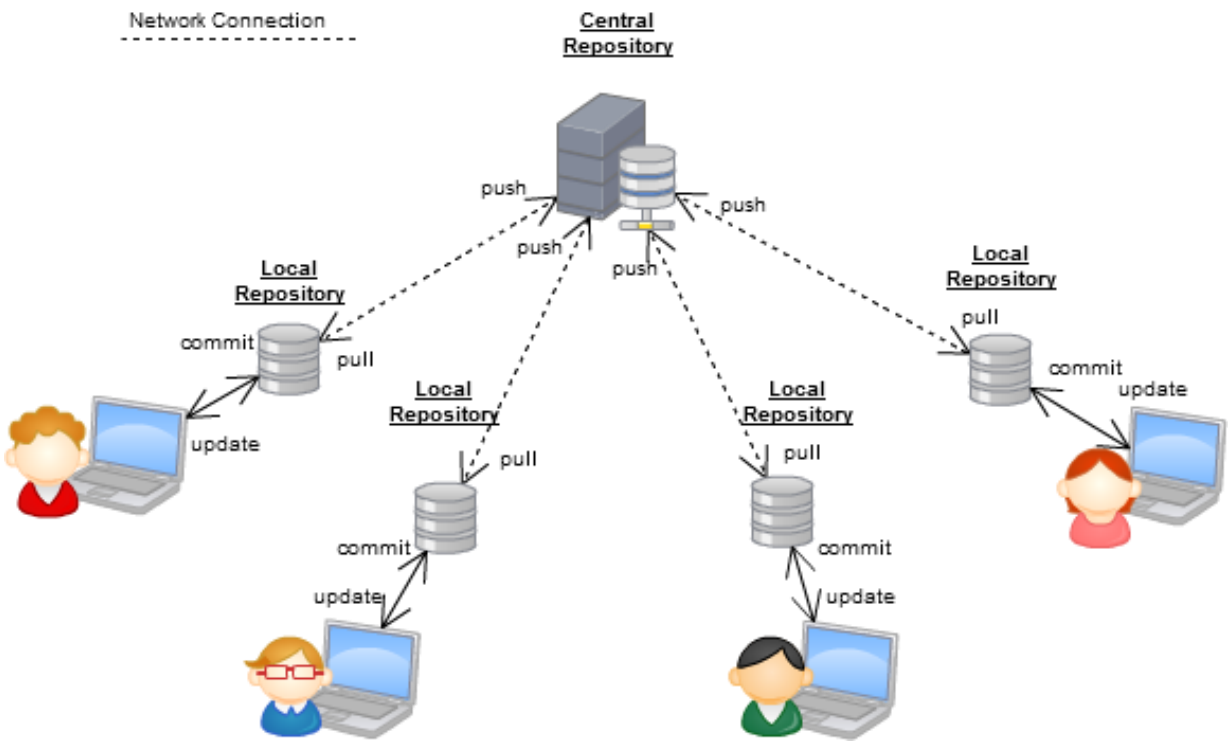
- Who made a change when.
- Collaboration tool (possibly reviewing)
- Rollback,
- When was a bug introduced
- Try out stuff without making backup 1 backup 2 etc.
- Best line based (**demo**)
- Not tied to any particular language
- Ownership

In the world of VCS git is predominant

- Many things have simplified over the years
- It's powerful for collaboration
- Be sure that others have solved problems you are facing
- It's made and used by massive distributed projects.
- It's simple enough for you to use
- It will make you happy

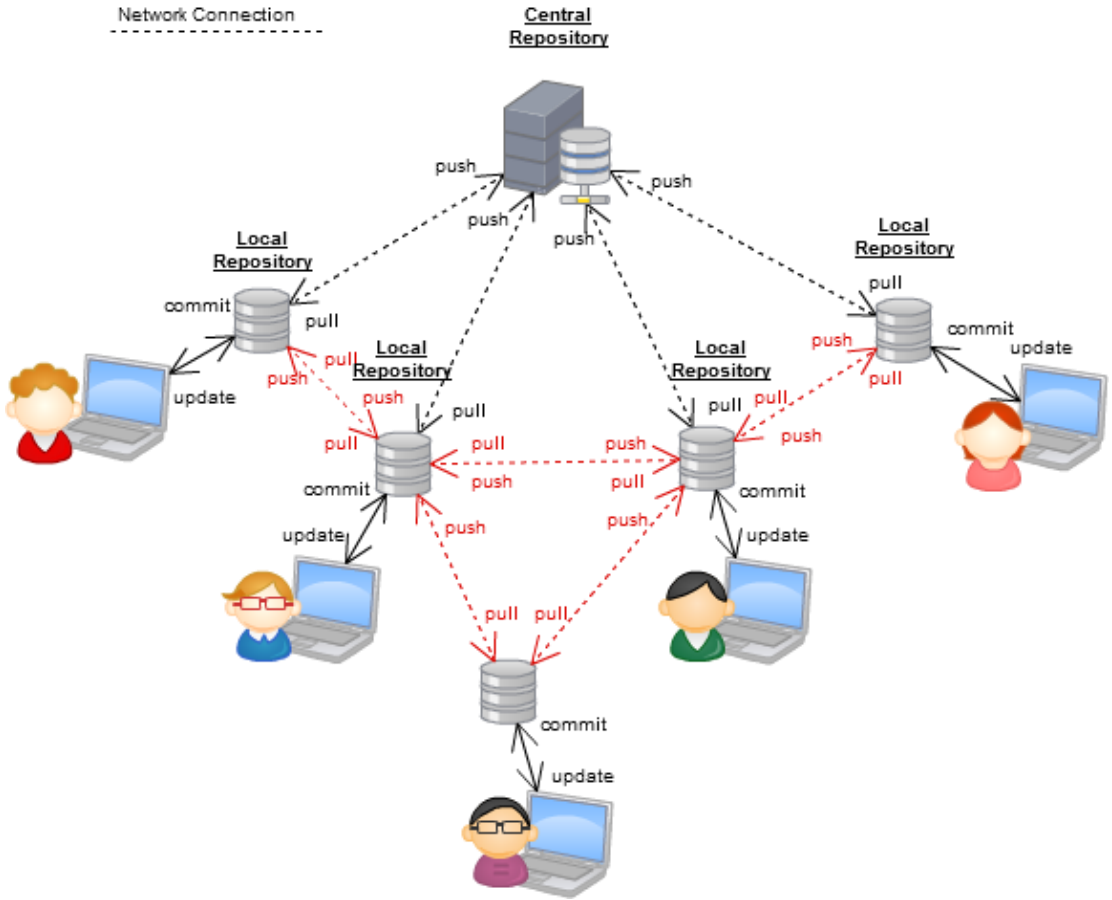
Git is decentralised (simplified)

- Git is fully distributed
- You operate on your local .git database until you push to server
- No central version numbers. Instead SHA-1 hashes (7a3b32f6a5b592b01542d9b36e14e4157d34b3d6)
- We have selected a central repository (TFS-server).
- Git repositories can be hosted "anywhere".
- Control is on repositories + whatever the hosting offers.



Git is decentralised

- Slide before was simplified
- Each copy is equally valid
- This is the concepts of remotes. Named servers
- Many hosting services (github, TFS, Stash, Git Lab, command line).



Exercise

Make your own local repos

```
➢ cd ~   #The home folder is a convenient location
➢ mkdir gorillashop
➢ cd gorillashop
➢ git init
```

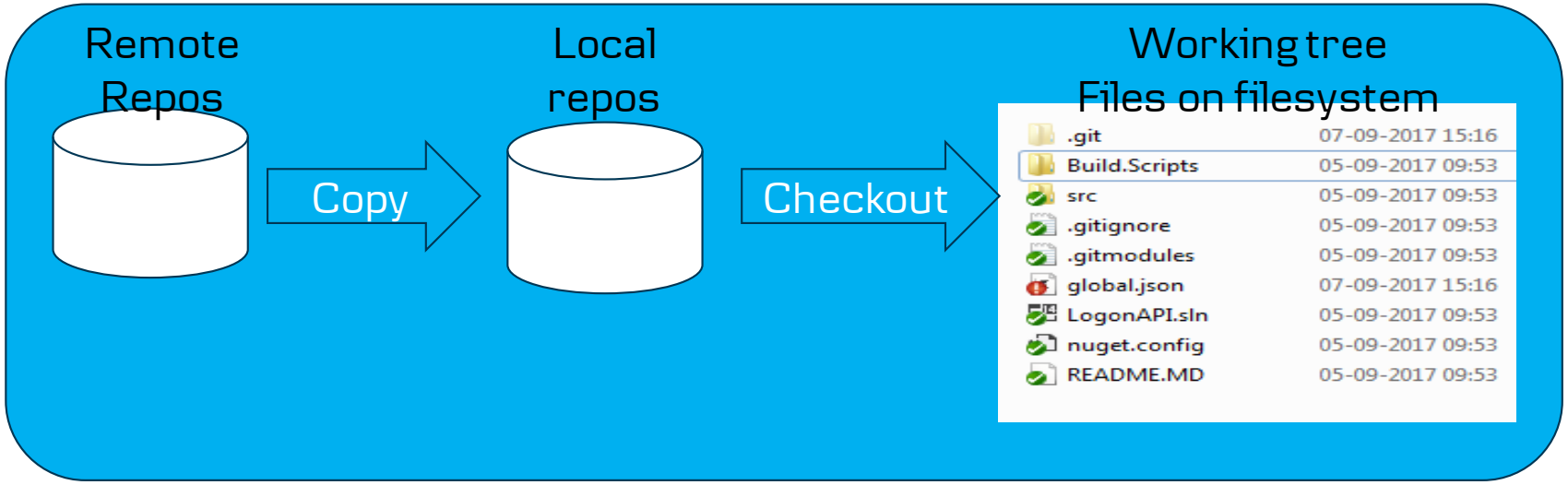
- This is handy when it's just you who want to commit. It's a full git repos
- We need some history so we will work on a repos that I have already made
- Will not use this repos in the coming slides

Exercise

Get a copy – git clone

- Some shared code
- Random hosting solution that every one can get.

git clone = initial copy to local repos AND checkout of files to filesystem




```
➢ cd ~   #The home folder is a convenient location
➢ git clone https://github.com/dstiel/marts2018.git
➢ cd marts2018
```

Exercise

Looking at the repos

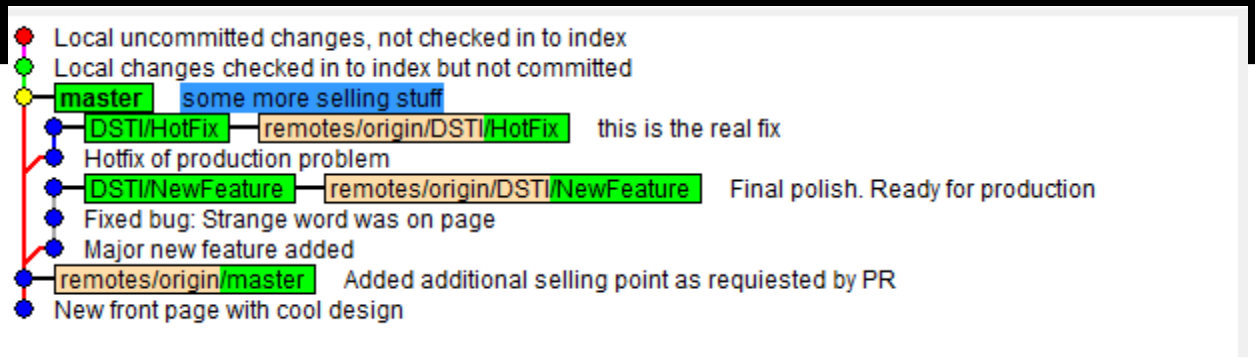
We will return to this when we have explained the concepts

- We see the remotes (what is **origin**)
- We see branches we se objects
- We see hooks
- We see a HEAD file

Lets try some git commands

- git log
- git log --name-only
- git log --pretty=oneline --graph --all
- git remote
- git remote get-url origin

(git instaweb) on some machines



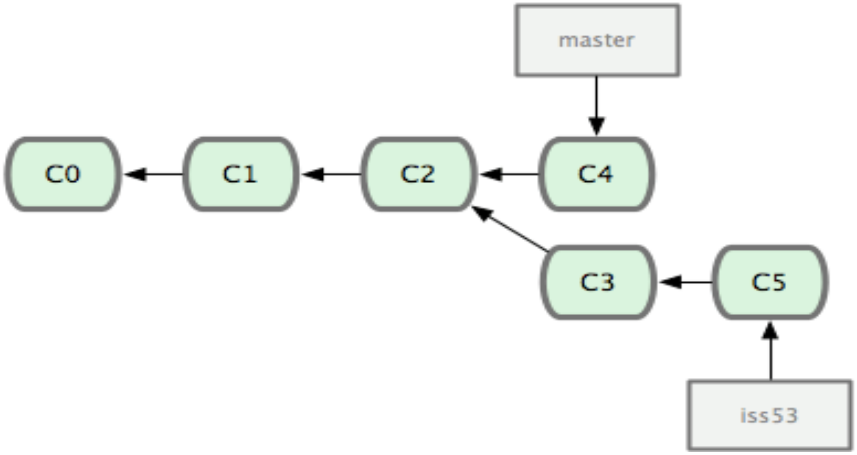
Commit objects

- Git stores a directed graph of **commit objects** (that points to their ancestor).

Each commit object has:

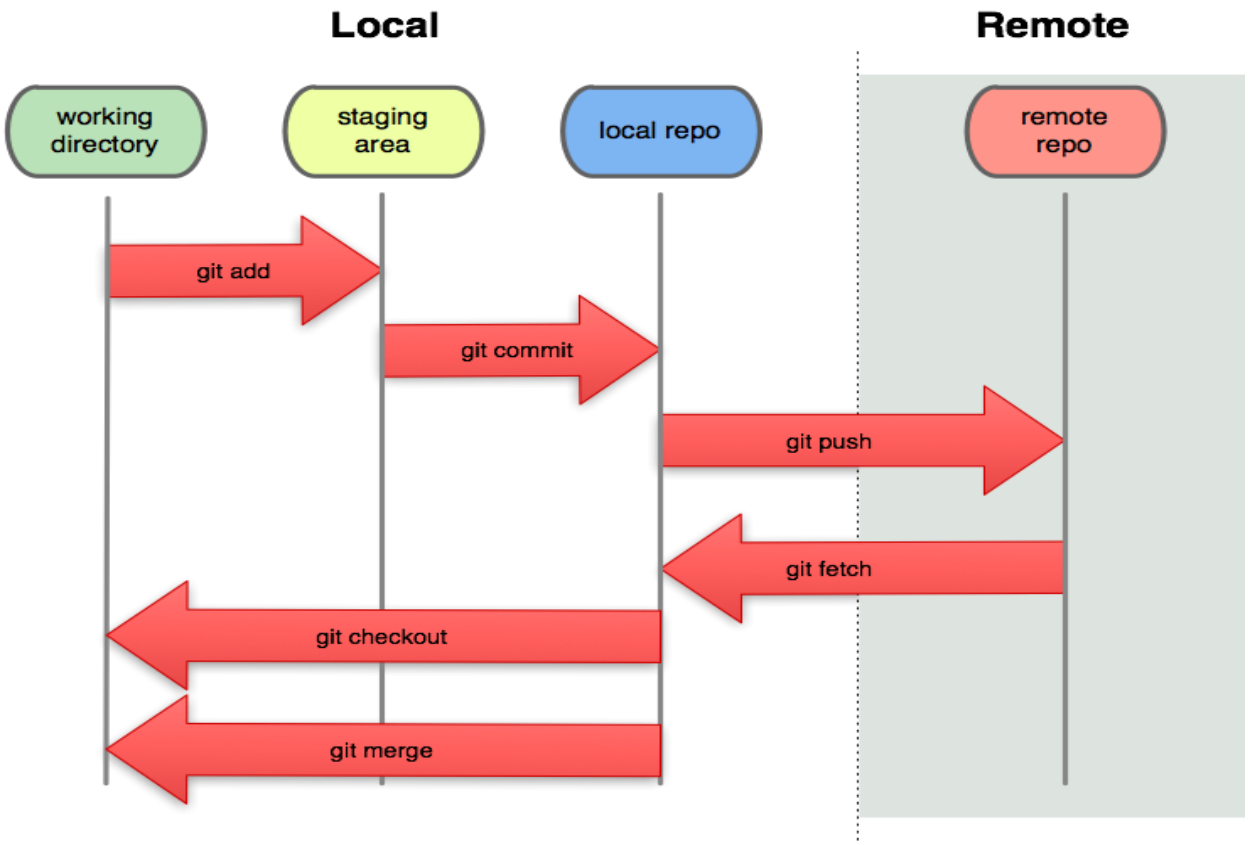
- SHA-1 code (a3b32f6a5b592b01542d9b36e14e4157d34b3d6)
- Contains pointer to ancestor
- Who made it
- A pointer to the tree (all the files in the commit)
- Commit message

```
➢ git log --pretty=oneline
➢ git cat-file -p fc4846e81
```



Making a commit

- Remember: You only commit locally
- Staging is a copy of the file

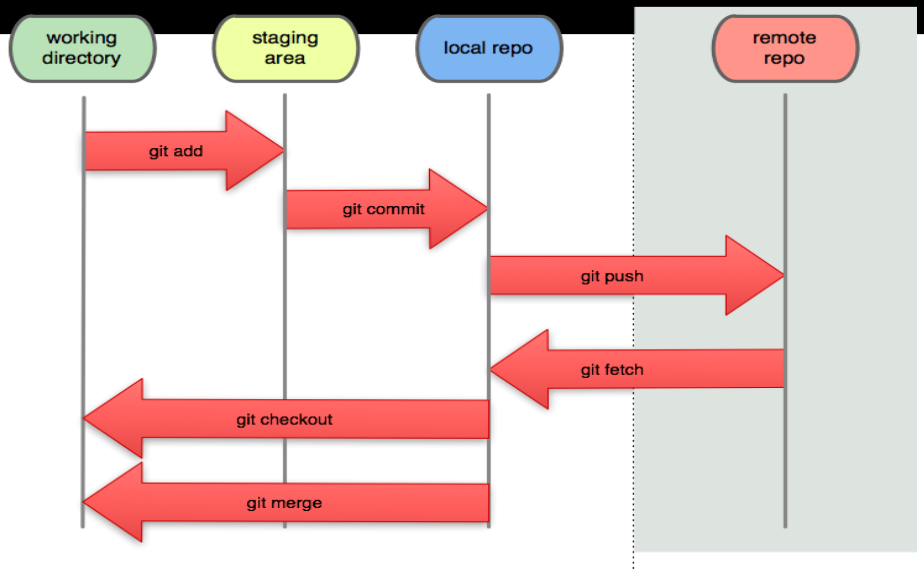


Exercise

Make a commit

- Change p1.html to contain a new
- We will get back to the push and the fetch/pull

- git status
- git diff
- git add p1.html
- git status
- git commit -m "Adding a line of code"
- Later git push..
- git log -pretty=oneline



Danske Bank

Exercise

Undo changes in file

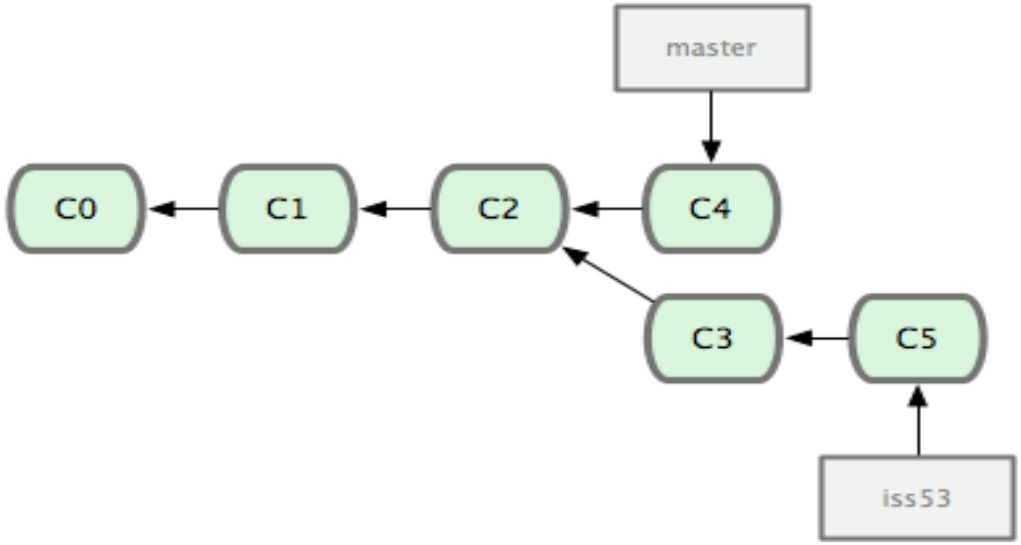
- Change p1.html to contain a new

```
➢ git status
➢ git checkout – p1.html #p1.html is now like it was
```

Branches –the MOST important slide

- Git stores a directed graph of commit objects (that points to their ancestor).
- Each commit object has a SHA-1 code (a3b32f6a5b592b01542d9b36e14e4157d34b3d6)
- A branch is a pointer that points to a commit in the tree.
- A branch pointer is moved when you make a new commit
- A branch is NOT a copy
- When checking out Branch the content in the working tree
- A branch is cheep (it's a pointer)
- Always draw this tree
- You can only commit to "local" branches, but they can be tied to a remote (upstreaming)
- You a "always" on a branch

A branch points to a commit
In the graph



Exercise

Looking at the branches -again

- git branch
- git branch -a

- Some branches are local, some are remote.

Exercise

Change branch

- There are several branches in what you cloned
- Difference between local and remote branches

- git branch –a
- git checkout origin/DSTI/HotFix #not good
- git checkout -b DSTI/HotFix origin/DSTI/HotFix #read what it says
- git branch –vv #see tracking/upstream branches

- Tracking branch / upstream got to do with push and pull. Ie. Getting and uploading changes to the remote

Now make a change and commit it on that branch

See what HEAD is pointing at now in the .git folder

Exercise

Make your own branch

- Change back to master branch and make a new branch from there
- Short cuts for many common tasks

```
➢ git checkout master
➢ git branch You/Feature
➢ git checkout You/Feature
➢ git branch
```

- Often you combine the branch and the checkout. Lets try that

```
➢ git checkout master
➢ git branch –d You/Feature #remove the branch again
➢ git checkout –b You/Feature #Create and checkout in one line
➢ git branch
```

- Now make a commit on the new branch
- Let's take a look at HEAD

Exercise

Changing branch –problems you will see

- Git won't let you loose changes unless you explicitly ask for it.

```
➢ git checkout master
Make a change in p1.html and commit it
➢ git checkout [branch from before]
Make a change in p1.html DONT commit it
```

What to do:

- Undo the changes in the files (read git status)
- Stash all changes

```
➢ git status
➢ git stash push
➢ git status
➢ git checkout master # now it worked

#to get your changes back –including conflicts
➢ git stash pop #we will get a conflict.
Remove the diff markes in p1.html
➢ git add p1.html
➢ git commit –m "stuff"
➢ git status
```

- Takes all changes in working tree and staging and put it in a stash
- It is a stack

Checkout can do two different things

- Can work on a file(s). This does not change HEAD, but only the content of a file (in working tree)
- Can work on a branch. This changes HEAD

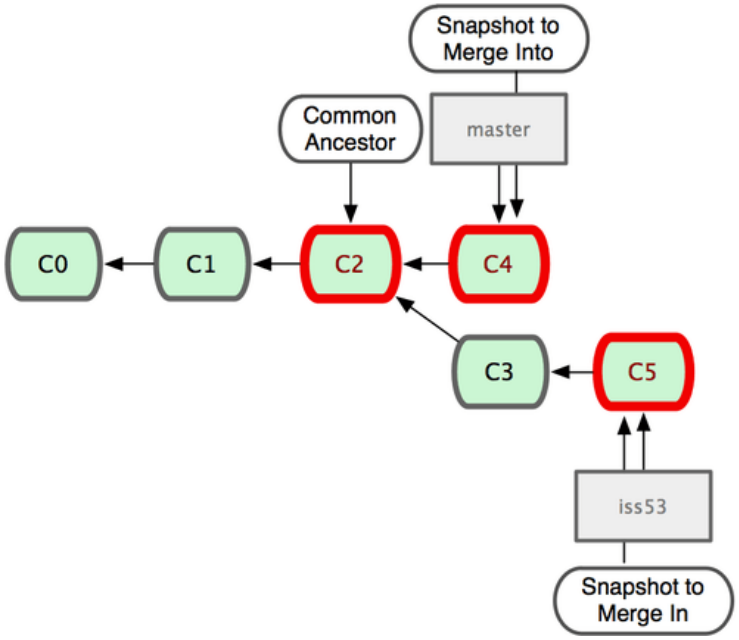
- git checkout DSTI/HotFix#HEAD changed

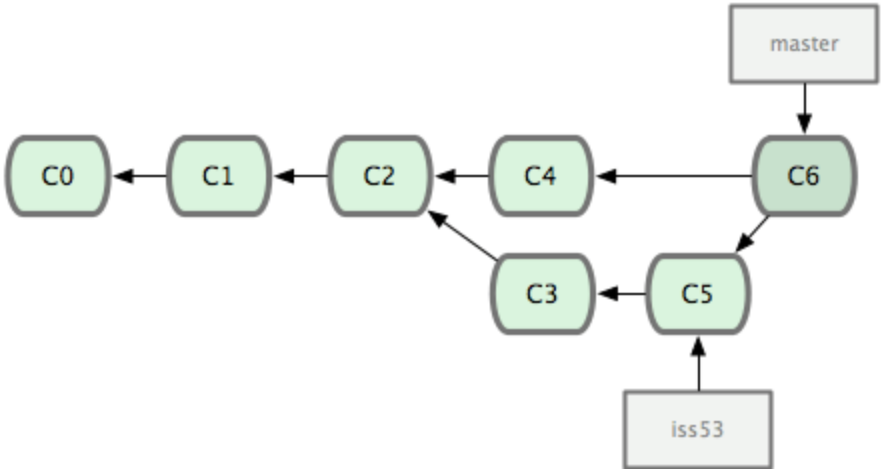
- git log
- git checkout HEAD~1 -- p1.html #changes the content of the file in the working tree and stage it. This was also ”undo” if you write HEAD or nothing
- git checkout SHA1 -- p1.html #changes the content of the file in the working tree

Merging

Now we can create and change branches. Now we need to combine the work again. Merging:

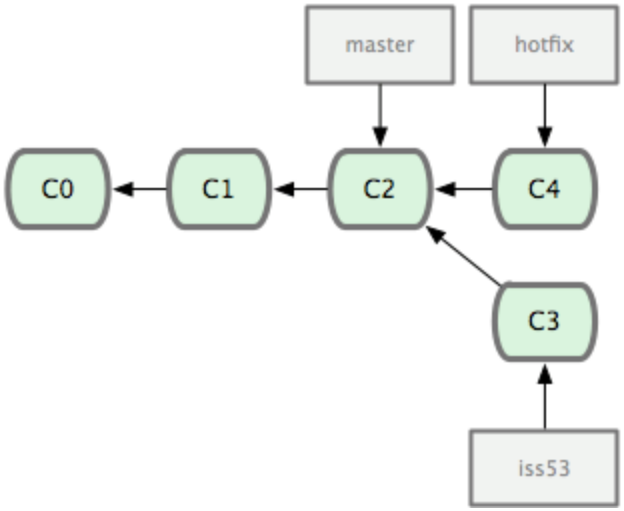
- Crawl back in the directed graph
- Commit object is created with two parents

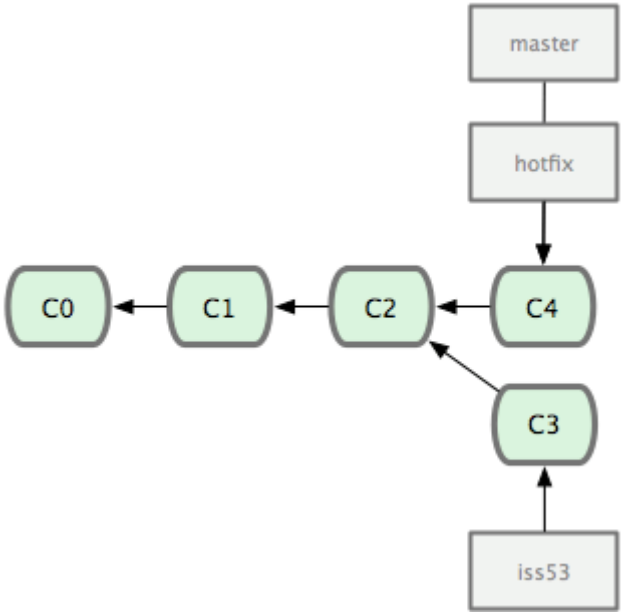




Merging –fast forward

- Sometimes you don't need to make a commit object to merge.
- Merge hotfix into master
- Nice linear history





Exercise

Merging branches -locally

- Remember we always do everything locally. Then push the changes.
- Remember others might have committed if you are sharing code (we will get back to that)
- You always merge into your current branch (HEAD)

```
➢ git checkout -b F2
Make a change to p1.html and commit it
Now merge it into master
➢ git chekout master #always merge into your current branch
See what is in p1.html
➢ git merge F2 #all is smooth
➢ git log --pretty=oneline --graph --all
```

Exercise

Merging branches –You can't harm your files

- Make a branch, C1, and make a change to p1.html in it.
- Change back to master branch.
- Make an almost identical change in p1.html but don't commit.
- ➢ Git merge C1
- Now commit p1.html to master

If we merge C1 into master we will get a conflict, because both have changed in

Exercise

Merging branches –resolving conflicts

- Conflicts happens when there are changes in the “same” locations
- Conflicts are natural and will occur.
- Conflicts are sometimes hard, sometimes simple.
- Different views/tools for resolving conflicts.

- Continues from last slide
- Now merge C1 into master. It will say conflict.

➢ git status
➢ Open p1.html. Fix diff markers
➢ git add p1.html
➢ git commit –m “merged in C1”

Caution. Complete a merge before implementing new stuff.

You can abandon a merge operation (with conflicts) before the commit

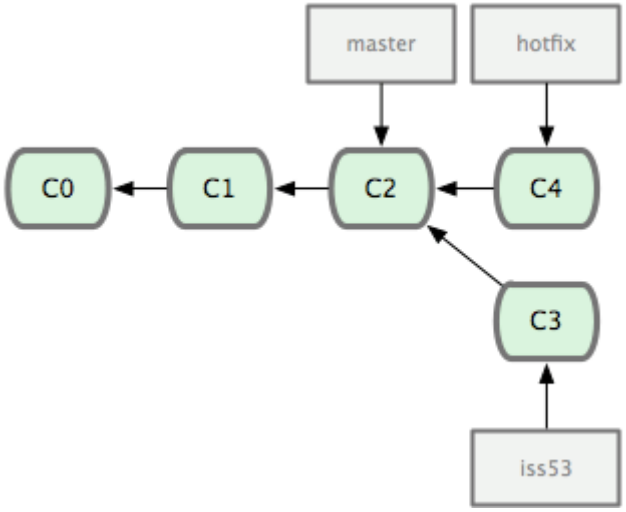
➢ git merge --abort

Exercise

Merging branches –what is already merged

- It's easy to see which branches have or have not been merged into the current branch (HEAD

- git branch -a --merge
- git branch –a --no-merged



Knowledge

Pull requests

Now we know how to create and merge branches. What about pull requests

- Request someone else to merge in your branch (in some repository) into a branch (on their repository). That what a pull request is for.
- A “raw” pull request
- Hosting services offers additional features, like commenting on the code, access control on branches etc. Making pull requests a popular way to collaborate and govern projects

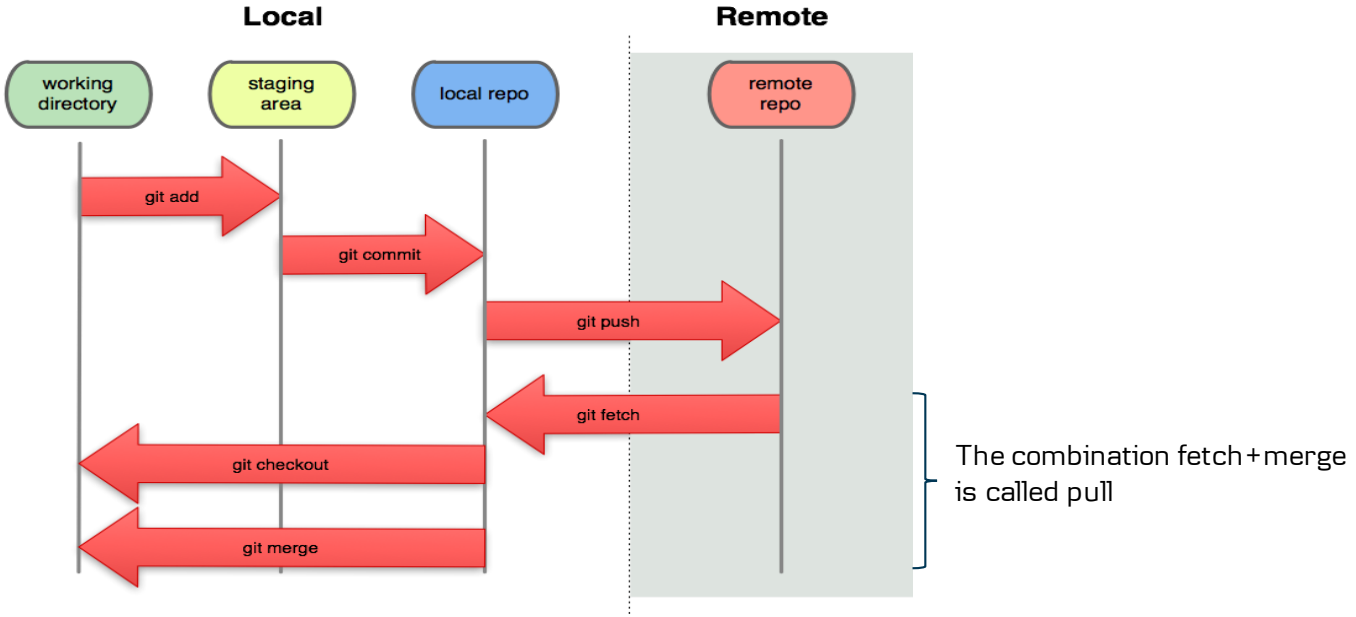
```
➢ git request-pull DSTI/HotFix origin master
#origin is a public url where DSTI/HotFix can be retrieved
```

Show TFS PR

Easy way to let people contribute, without giving away control

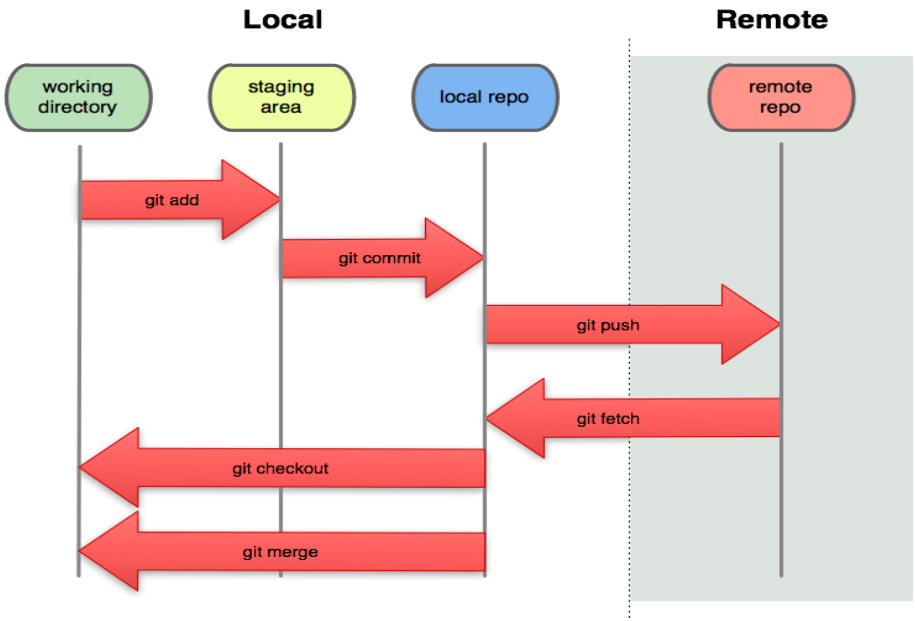
Working with remote servers

- Everything we have done has been done locally.
- If you want to share your work with others the git repository must be accessible to them
- Working with remotes is all about pushing your database into theirs. And merging theirs into your.
- This is called *push* and *fetch*
- Things can have changed in the remote repo since your last fetch



Working with remote servers

- Tracking branches are convenient because you can push and pull.
- If you work with branches origin/master to your branch often.
- Remember others can't see your work until it's on a remote server



Danske Bank

Exercise

Git pull –warm up for the next

Demonstrate tracking branches

- cd ~ #The home folder is a convenient location
- git clone https://github.com/dstiel/marts2018.git marts2018-2
- cd marts2018-2

Modify the first <li> in p1.html to contain a new text
Commit the change (git add followed by git commit)

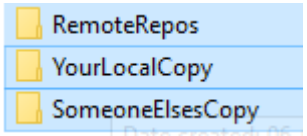
Now I will make a change on the server to p1.html

- git fetch
- git merge origin/master

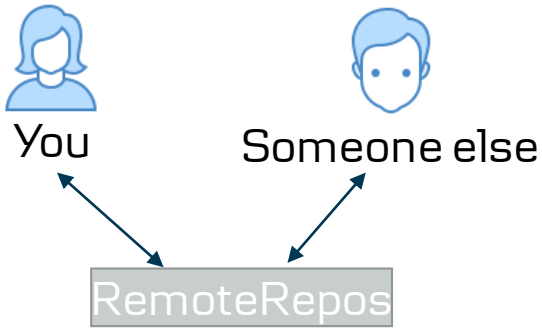
The two can be combined into a pull.
You will get a conflict.

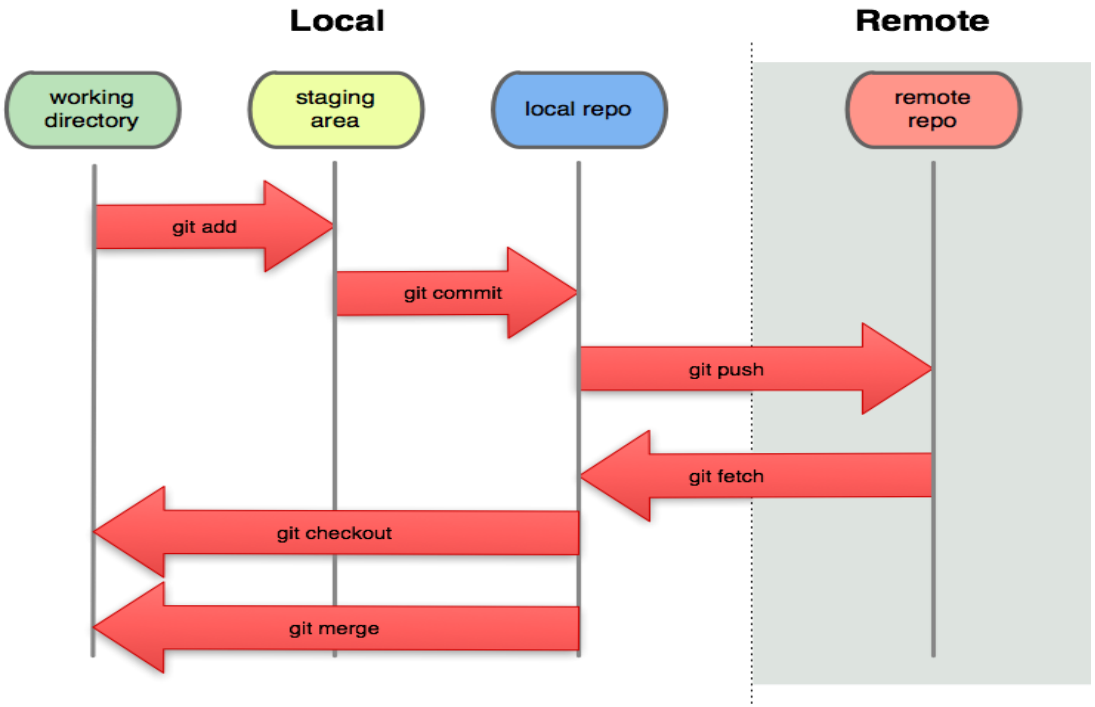
Resolve the conflict

The "final" Exercise.



- Start all over
- Make following folder structure
- Hosting it your self, or hosting services.
- Sharing a folder is also enough.
- We will use a local folder as the "remote server".
- All commands are identical. Only the URL is different.





Exercise

The "final" Exercise.

Setup a repos in RemoteRepos
- Make a subfolder Project2
- In Project2 initiate a new git repos

In YourLocalCopy
- Clone the git repository Project2 from RemoteRepos (git clone ~/RemoteRepos/Project2
- Make a file p2.html and that looks like p1.html
- Commit the file
- See what origin is
- Write "*git push" to push it to the RemoteRepos*

In RemoteRepos
- Clone the git repository Project2 from RemoteRepos
- See what you get
- Add file. Push.

Exercise

Creating new remote branches

In YourLocalCopy

- Make a new branch C2
- ➢ git push # doesn't create the branch remotley
- ➢ git push HEAD origin -u #creates the branch and tracks/upstreams it it
- Make a change in a file
- Commit the change
- Push the change

In SomeoneElses check when you can see the branch

- ➢ git fetch
- ➢ git branch –a
- When you can see C2 you should create a local branch from C2(change branch slide)

Tracking/upstreaming branches is the feature where push and pull works easily.
If you forget to configure the upstream branch you can always do.

Git branch –u origin/branchToTrack

Danske Bank

Complete change of scene. Random Git stuff

By
David Stiel
dsti@danskebank.dk

Exercise

"Undo" commits – git reset

What do you do when you forgot a file or wrote a really bad commit message…? Rewrite history. Move your pointers

- Make a change in p1.html
- Commit the change
- ➢ git status
- ➢ git reset HEAD~1
- ➢ git status

What does it do:

- Move the branch pointer (in HEAD) back one (or more) commit.
- Doesn't change the files. So you can make a new corrected commit
- Doesn't work if you have shared your commits.

Exercise

Reset your repository to a previous state

You have developed something but regret and want to clear your working tree.

- Make a change and commit it
- Add a file and commit it

```
➢ git reset HEAD~2
➢ git clear
```

```
➢ git checkout origin/master
➢ git clear
```

Knowledge

Submodules

- Your program depends on other independent libraries. How do you “embed” their source code in a versioned and easy to modify way.
- You and your team should be confident in git. But then it works
- Maybe start with some simple scripts
- It embeds a fully valid git repository within your git repository. And makes a checkout at a particular version (detached head).

Exercise

References

- https://git-scm.com/
- Pro Git book (available from git-scm or printed)