Danske Bank

Introduction to Git

David Stiel
Danske Bank
dsti@danskebank.dk

Danske Bank

Prerequisites

- Some Git
- Git from Command line
- An editor

Goals

- Understand the concepts and terminology and what goes on inside
- Understand everyday git tasks
- More insights (maybe too much) than most people have

Danske Bank

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.

Quiz

What is git?

- A version control system
- A backward directed graph of something
- A key-value store

Knowledge

What is version control systems (VCS)

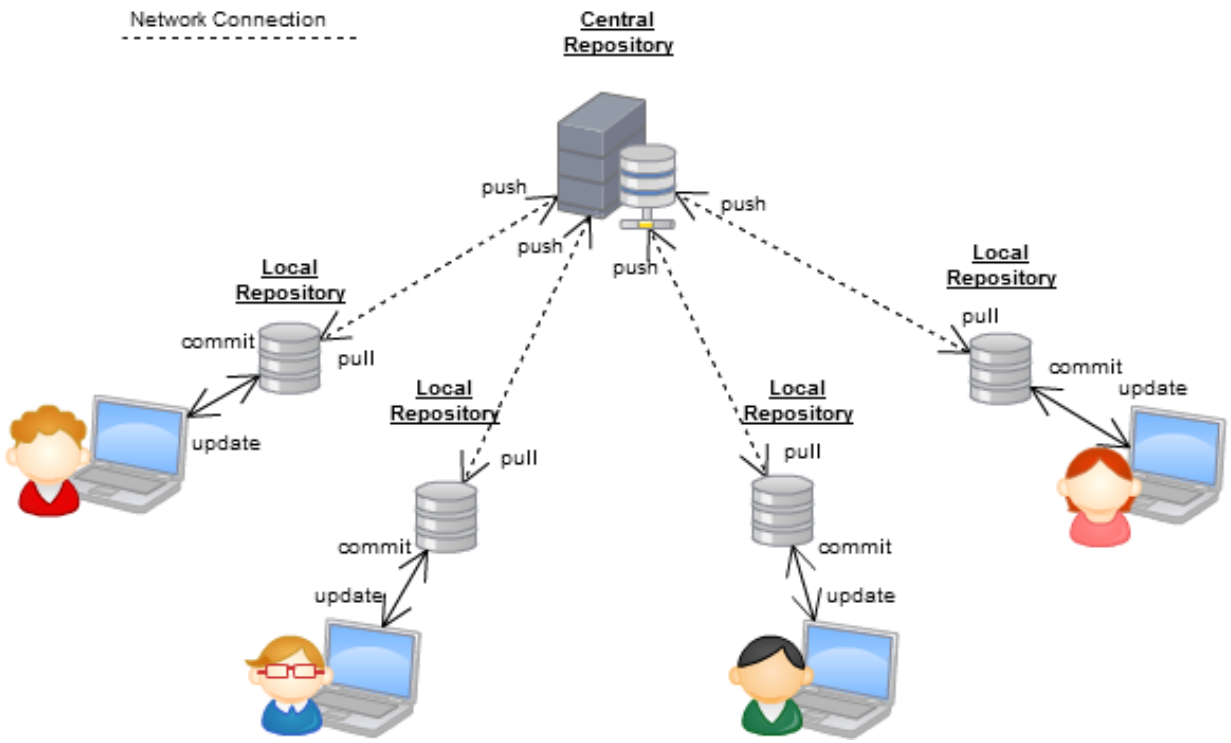
- Who made a change when.
- Collaboration tool (possibly reviewing)
- Rollback,
- When was a bug introduced
- Try out stuff without making backup 1 backup 2 etc.
- Best line based (**demo**)
- Not tied to any particular language
- Ownership

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

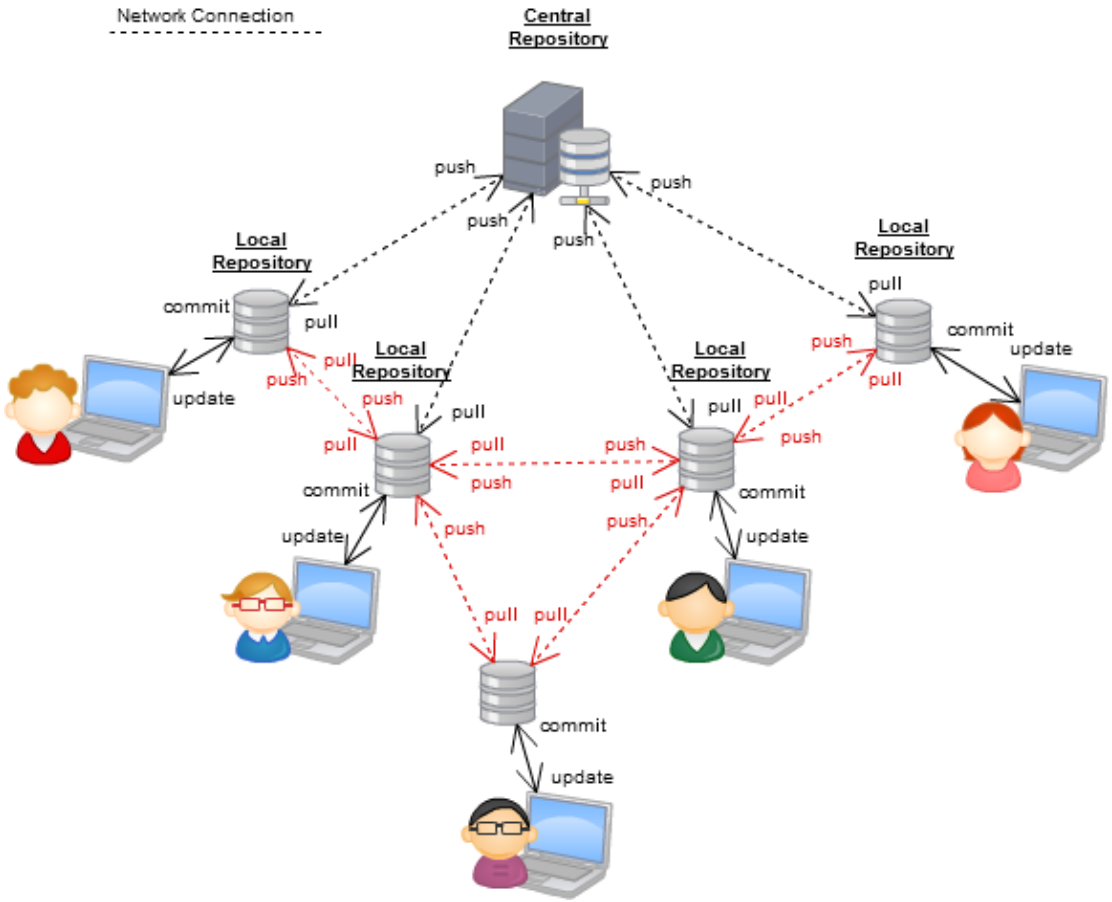
Git is decentralised (simplified)

- 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)
- 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).



Danske Bank

Exercise

Make a local repos

We will use this for some exercises

- 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

- Make a tmp folder somewhere convenient #say c:\tmp or ~/tmp
- In the tmp folder
- mkdir gorillashop
- cd gorillashop
- git init

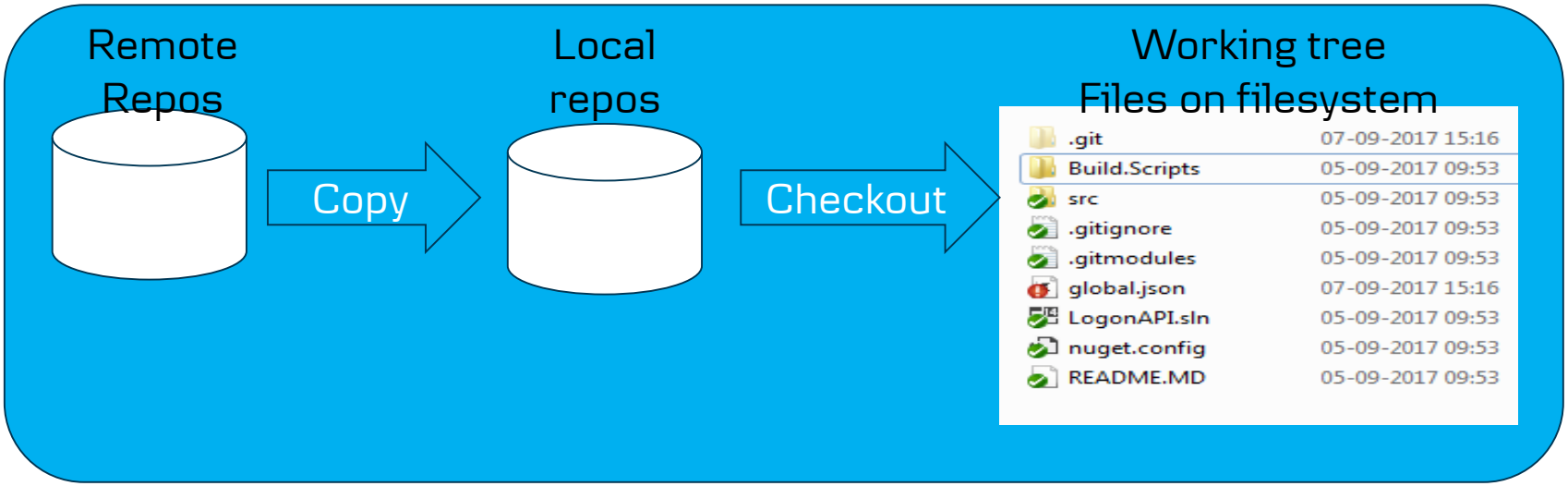
How many can add and commit a file to this repo?

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



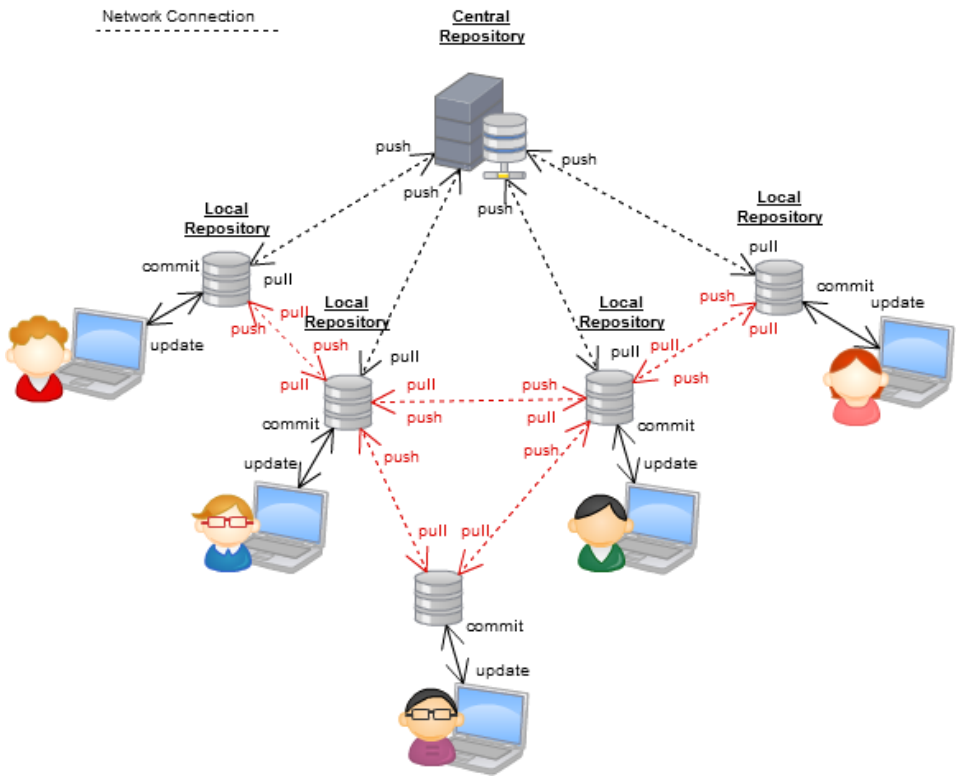
- cd c:\tmp #or ~/tmp
- git clone https://github.com/dstiel/marts2018.git
- cd marts2018
- Look in .git

Exercise

Looking at the .git folder

We will return to this when we have explained the concepts

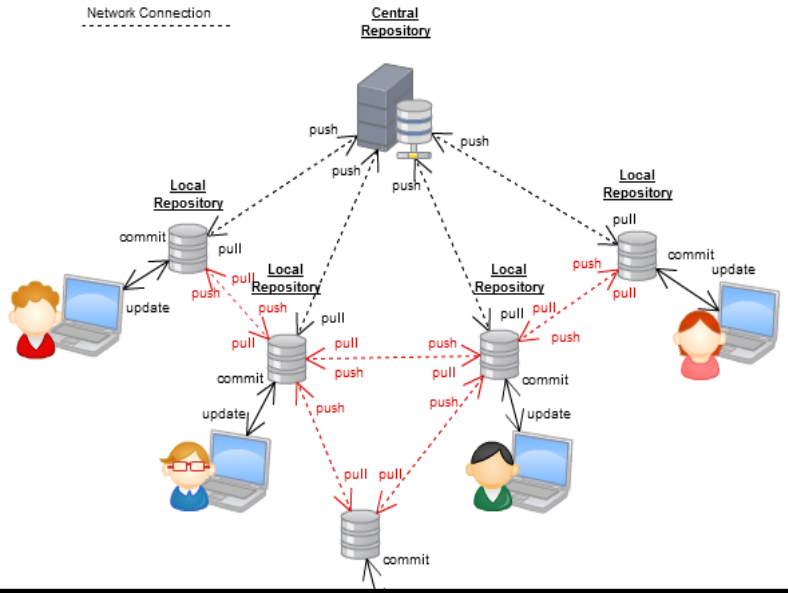
- We see the remotes (what is **origin**)
- Look in config
- We see se objects (the key-value thing) “git cat-file –p folder+filename”
- We see refs
- We see hooks
- We see a HEAD file



Knowledge

Git remotes

- Demystify remotes and origin
- Normally push and pull are just to origin
- /remotes/origin/master ?



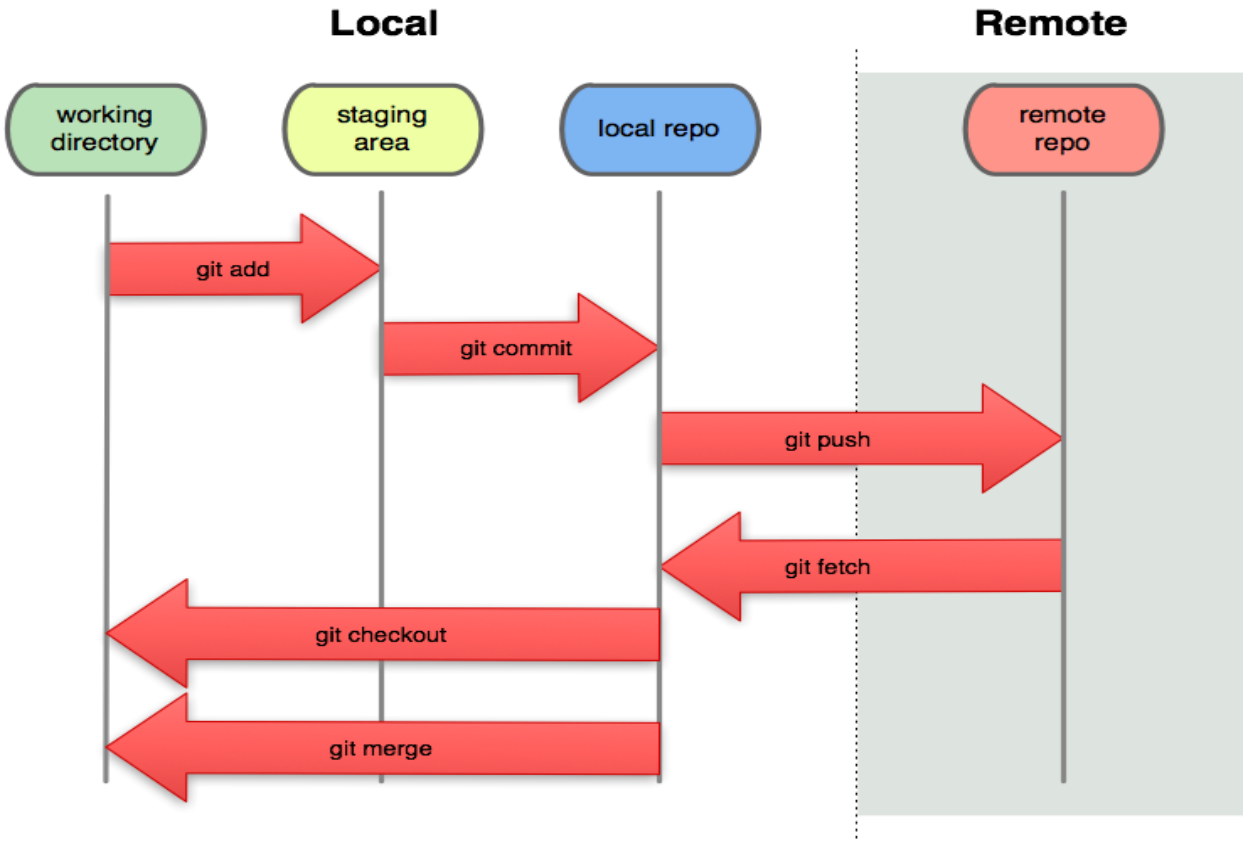
- In what you just cloned
- git remote
- git remote –h
- git remote get-url origin #this will show where where you initial clone came from

Add a remote (just for fun)

- git remote add gorilla c:\tmp\gorillashop
- See that you now have two remotes and check the url of the new one
- Look in the .git folder in the config. Do you see it
- git fetch gorilla
- Look in refs/remotes

Making a commit

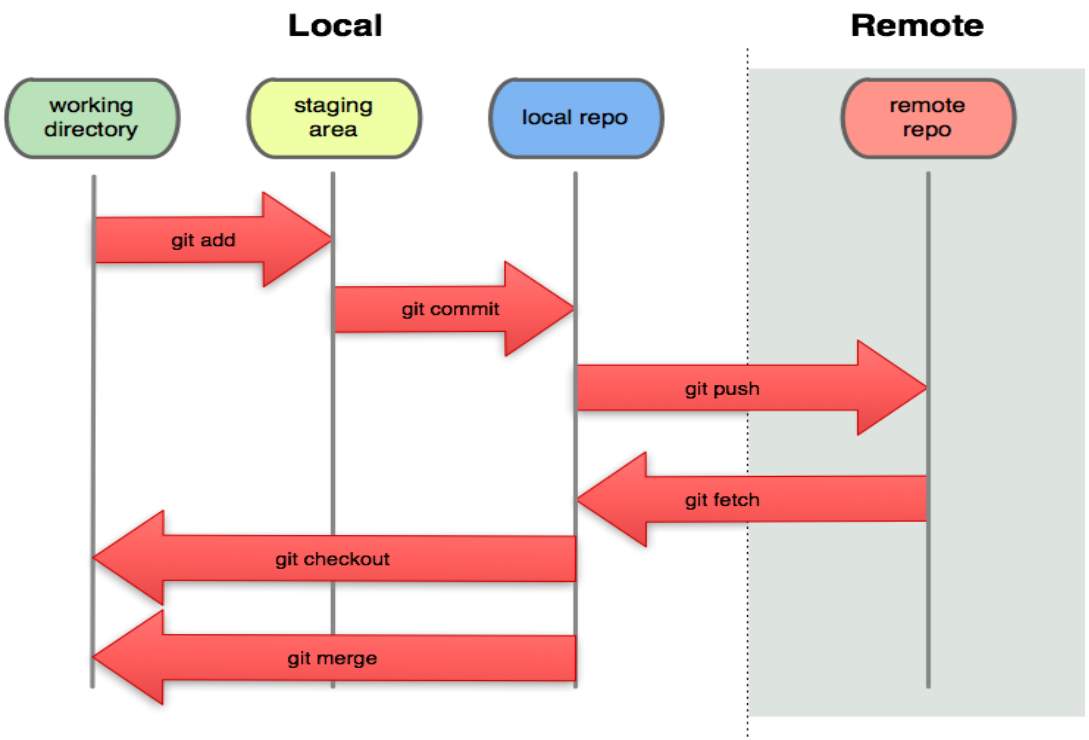
- You only commit locally
- A commit creates a commit object and moves HEAD to point to that
- Staging is a copy of the file.
- Staging area is also called index.



Exercise

Make a commit

- Change p1.html (c:\tmp\marts2018)
- git status
- git diff
- git add p1.html
- git commit –m “Your commit message”



Exercise

Look at the logs

- git log
- git log --pretty=oneline --graph --all
- git log --oneline --graph

A small tip

- git config --global alias.graph 'log --pretty=oneline --graph‘
- git graph

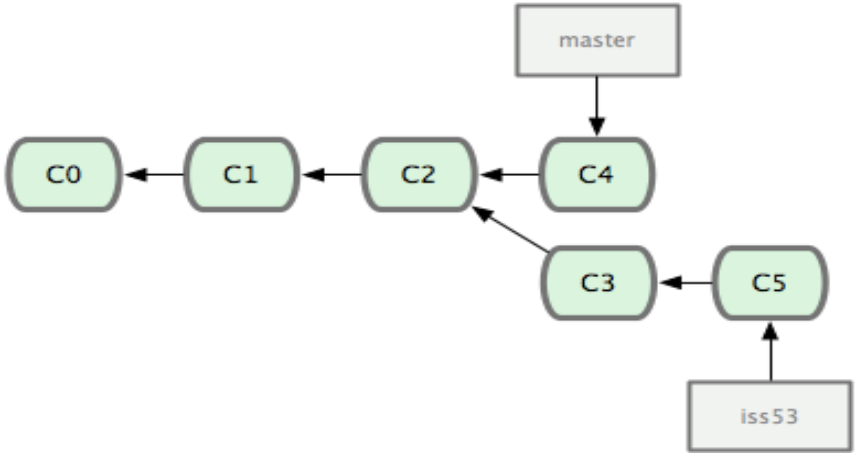
Commit objects

Git stores 3 kinds of files/objects. The filename is the sha1 of the content

- Commit objects
- Tree objects
- File/blob objects

Each commit has a commit object :

- SHA-1 code (a3b32f6a5b592b01542d9b36e14e4157d34b3d6)
- A reference to the ancestor(s)
- A reference to a tree object
- Who made it
- Commit message



Danske Bank

Exercise

Explore the objects (you normally never need to do this)

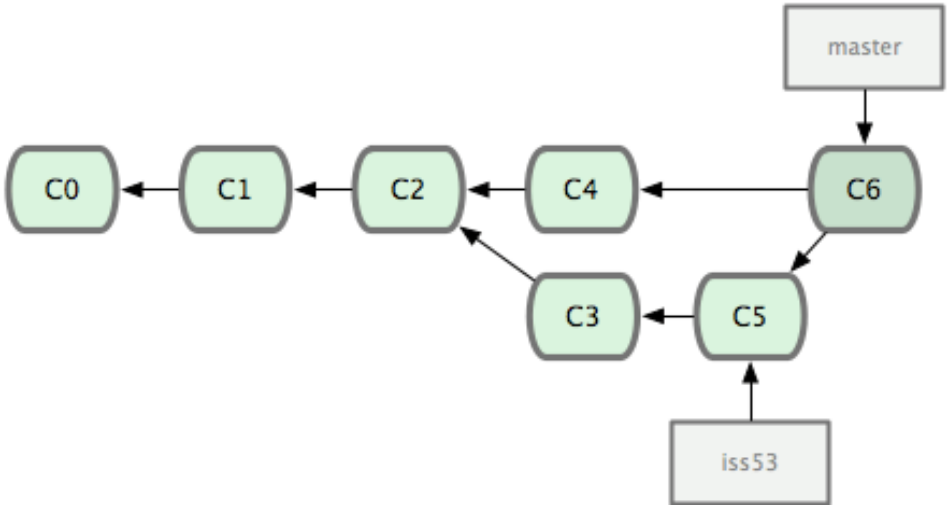
- Look at HEAD
- Add a file, commit it and make a fun commit message
- Look at HEAD
- git log --oneline (or –pretty=oneline)
- git cat-file –p [the sha1 of the commit / first couple of characters is enough]
- git cat-file –p [the sha1 of the tree from the commit object]
- git cat-file –p [the sha1 of the p1.html]

Merges sometimes creates a commit object with two parents

- git cat-file -p 8bb1afe

The important things to remember

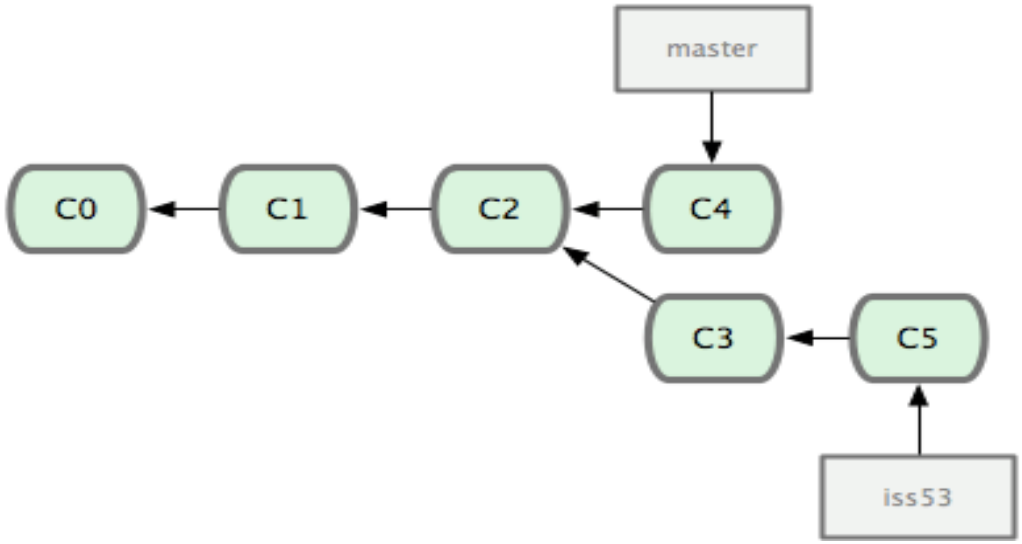
- Certainly not all the details above
- Each commit creates a commit object
- Commit objects form a (backwards) directed graph of all commits
- Git is about navigating this graph
- Merges sometimes create a commit object with two parents
- A commit object is identified by it's sha1



Branches –the MOST important slide

- Git is about navigating the tree of commit objects. Branches helps with that
- 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

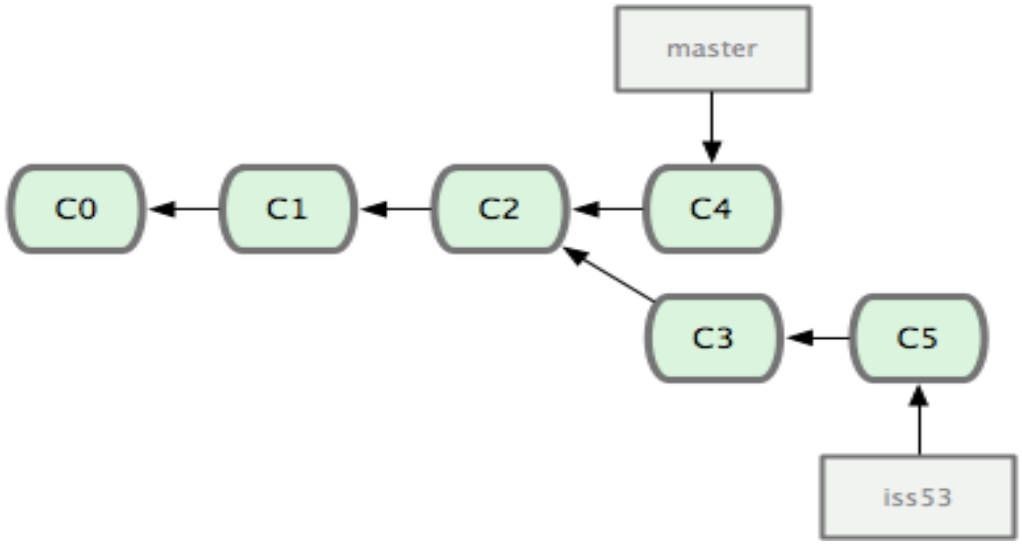


Branches

- Even if you don't branch **you do** when you push/pull.
- origin/master and master from refs folder.
- Push and pull is also branching
- We will look at creating and changing

1. Change branch
2. Making branches
3. Changing branches –problems, like you see with pull
4. Merging branches (this also happens with push and pull)
5. Push and pull

A branch points to a commit
In the graph

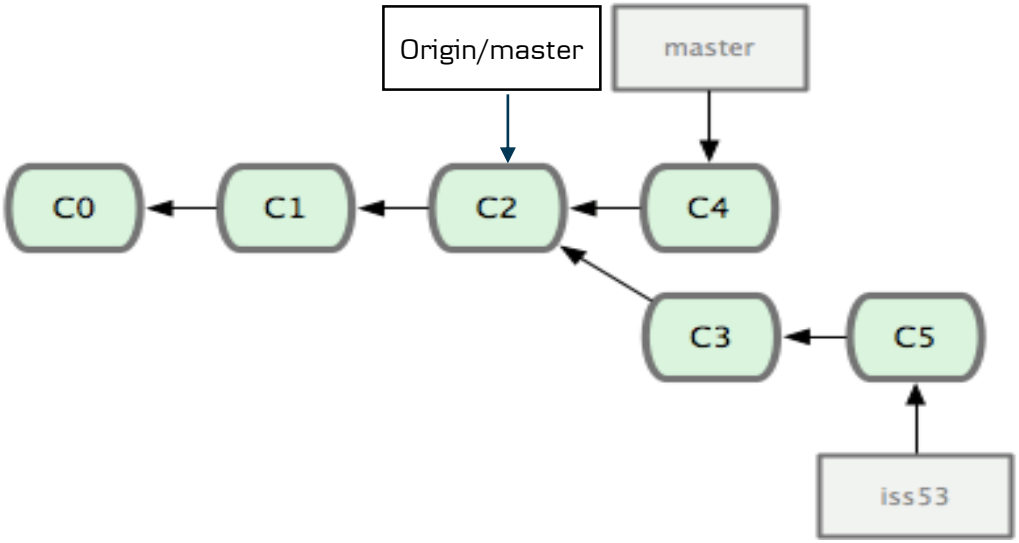


Exercise

Looking at the branches -again

- In marts2018
- git branch
- git branch –a
- git branch -vv

- Some branches are local, some are remote.
- Can only work on local branches
- Can tie a local branch to a remote branch –upstream. Matters for push and pull

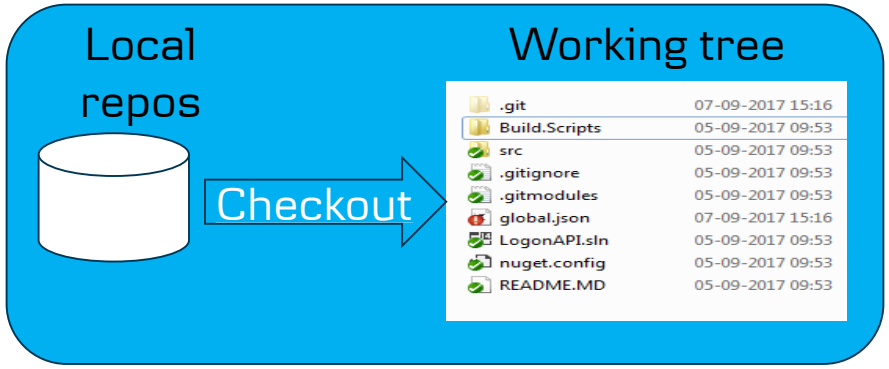


"git checkout" to change branch

Before going into branches let us look at checkout

git checkout takes "things" from the repos and changes the working tree.

- Checkout 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 and the working tree.



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

Make a new repo
- mkdir funWithBranches
- cd funWithBranches
- git init
- Make a file a.txt with some text in it and commit it
- git log --oneline --graph

Make an change branch
- git branch newFeature
- git branch #see you are not on the new branch
- git checkout newFeature
- git branch #you a now on the new branch

Often you combine the branch and the checkout (shortcuts). Lets try that
- git checkout master
- git branch –d newFeature #remove the branch again
- git checkout –b newFeature2 #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.

- Make a change in a.txt and commit it
- git checkout master
- Make a new line in a.txt with the text "Im sure this will conflict. Stash". DON'T commit it
- git branch
- git checkout newFeature2 #this should fail

What to do:

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

- git status
- git stash push #What does stash do?
- git status
- git checkout newFeature2 # now it worked

#to get your changes back –including conflicts

- git stash pop #we will get a conflict.

Remove the diff markes in a.txt

- git add a.txt
- git commit –m "stuff"
- git status

Exercise

Undo changes in file

- Change a.txt to

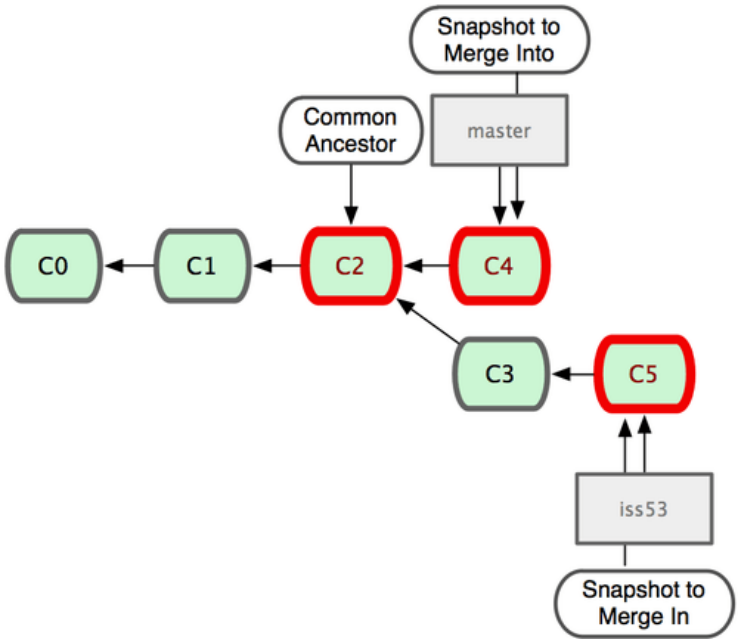
```
➢ Change a.txt
➢ git status #read it
➢ git checkout -- a.txt # a.txt is now like it was
➢ git reset  --hard #undo all your changes to files that are in git
➢ Make a new file b.txt
➢ git status
➢ git clean –f #remove all files not in git, eg. builds
```

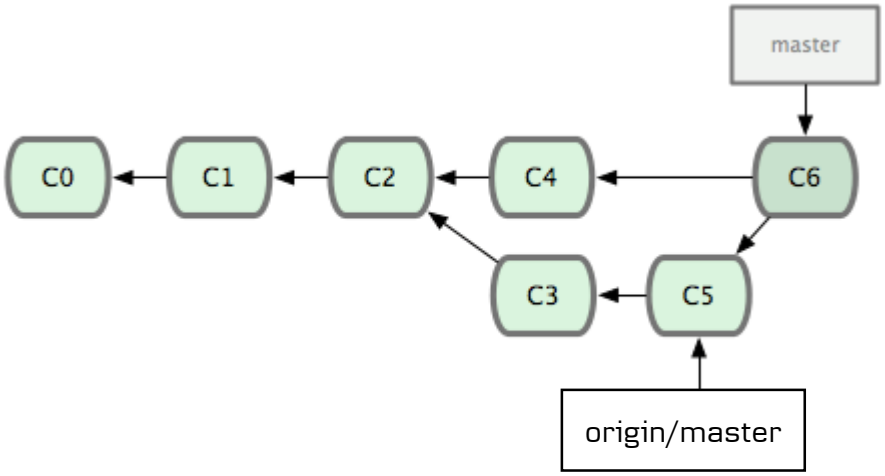
Merging

We can create an change branches
What to use branches for
Why you need to know if if you don't branch

After work in branch. We need to combine the work again. Merging:

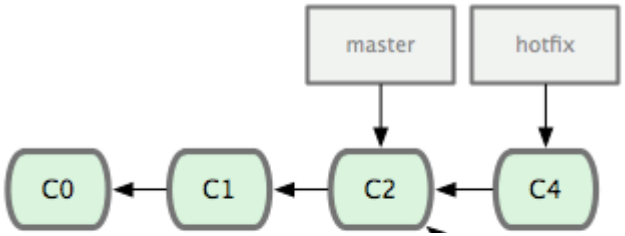
- Crawl back in the directed graph
- Commit object is created with two parents

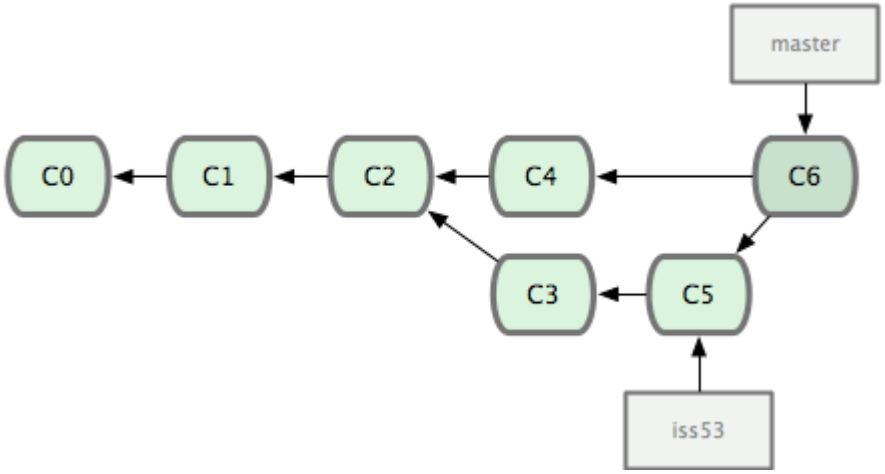




Merging two kinds

- Sometimes you don't need to make a commit object to merge.
- Sometimes you can just move the pointer
- 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)

First a fast forward

- Make folder and git init
- Create and commit a.txt
- git checkout –b ffBranch
- Modify and commit a.txt
- git checkout master
- git merge ffBranch

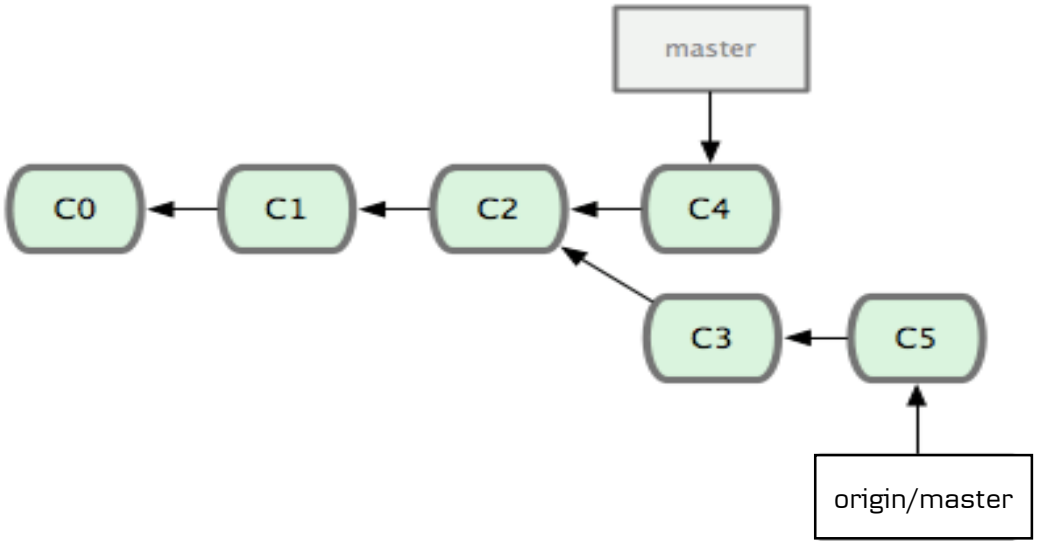
Merge commit

- git log --pretty=oneline --graph --all
- Make a change to a.txt and commit it (on master)
- git checkout ffBranch #write git branch if you forgot the name)
- Make a file b.txt and commit it (avoid conflicts)
- git log --oneline --graph --all
- **git merge master**
- git log --oneline --graph --all

That was only locally -boring

➢ git branch -a

- It's the same for remote branches, like mergeing in the work of others



Knowledge

Merging branches –is it dangerous

Git won't let you stupid things –unless you insist

Knowledge

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.

Caution. Complete a merge before implementing new stuff.

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

```
➢ git merge --abort
```

Danske Bank

Exercise

Merging branches –resolving conflicts

Make the branches to the same
- git log --oneline --graph --all
- git checkout master
- git merge ffBranch
- git log --oneline --graph --all

Make changes in the same file
- Add a line at the bottom of a.txt and commit
- git checkout ffBranch
- Add a line at the bottom of a.txt and commit
- git merge master

We will now get a conflict when trying to merge

Open a.txt
- Fix diff markers
- git add
- git commit

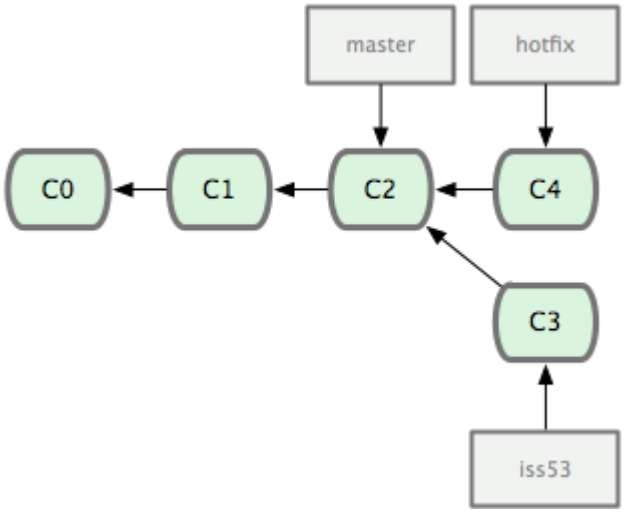
Caution. Complete a merge before implementing new stuff.

Exercise

Merging branches –what is already merged

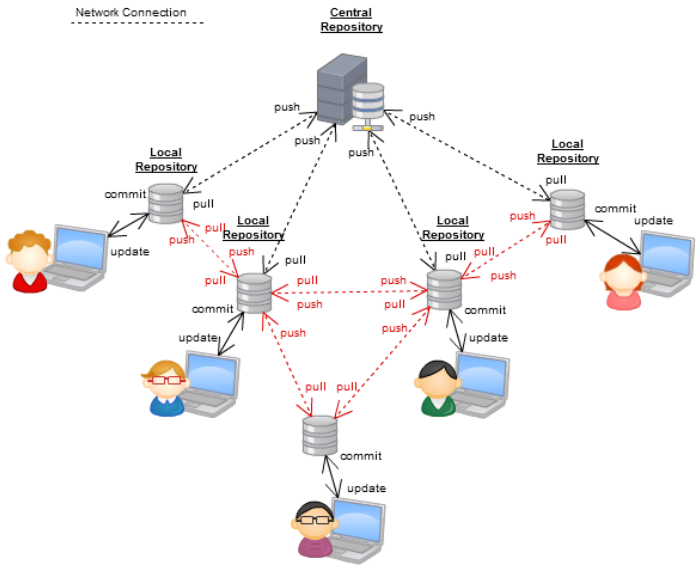
- It's easy to see which branches have or have not been merged into the current branch (HEAD

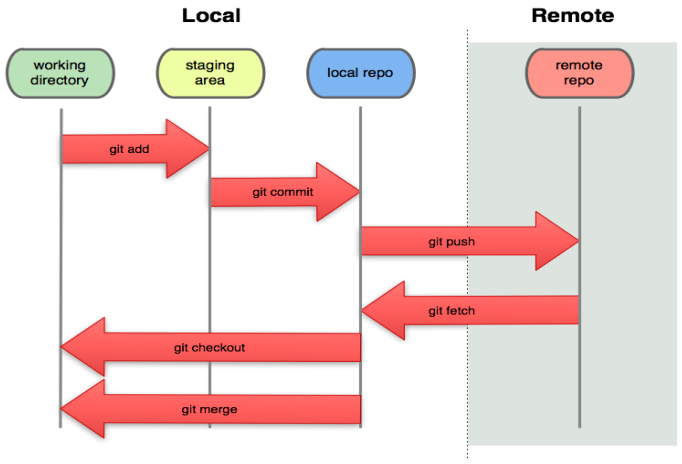
➢ git branch -a --merge
➢ git branch –a --no-merged



Pushing-share you code

- 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.





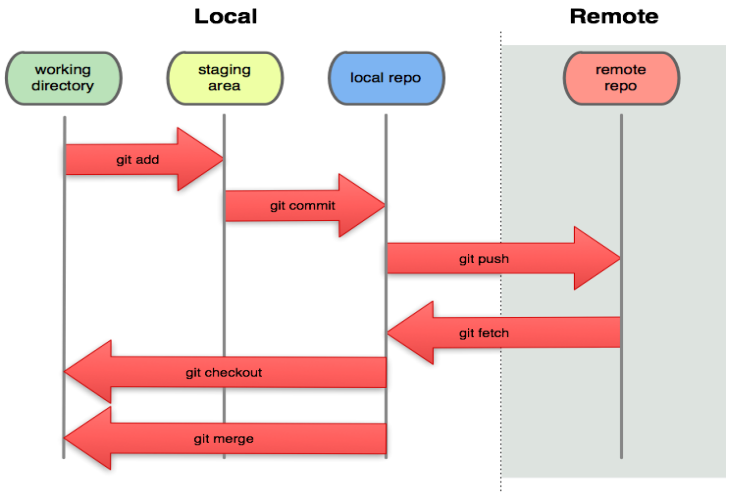
The combination fetch+merge is called **pull**

Pushing a branch to a remote

- If you have created a local branch you must first push it to the Remote
- If you already have an upstream configured push will update the remote branch and push objects.
- The remote might have rules stopping you from updating directly. Eg. Pull requests or access control.
- Remotes wont let you overwrite others people work, so you have to pull first

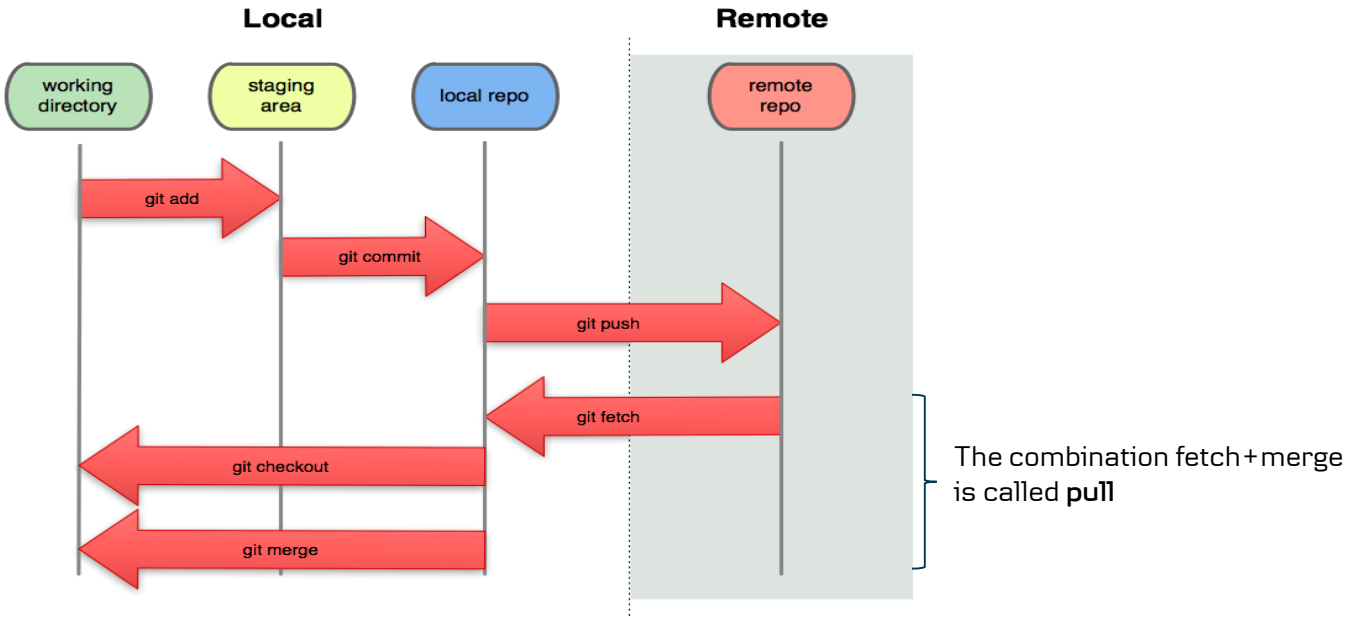
- git checkout master
- git checkout –b fancyBranc
- git push origin HEAD:[name on remote branch] **–u** #u for upstream
- git branch -vv

Now you can push and pull

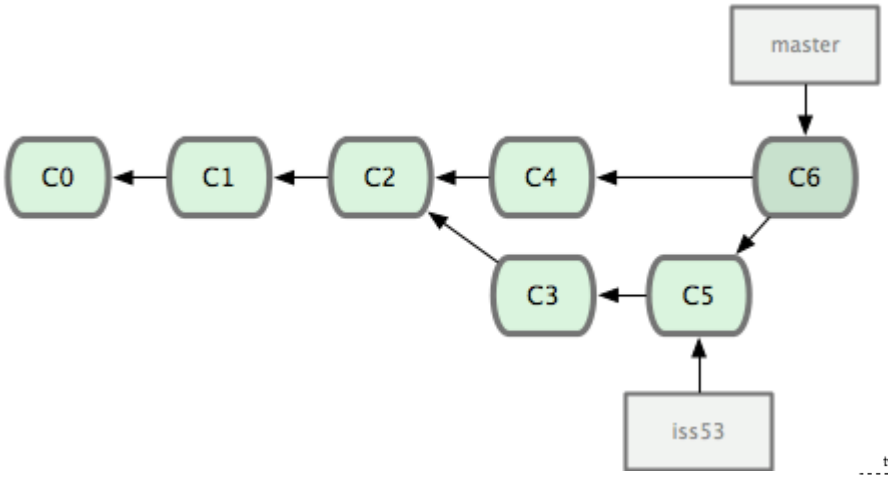


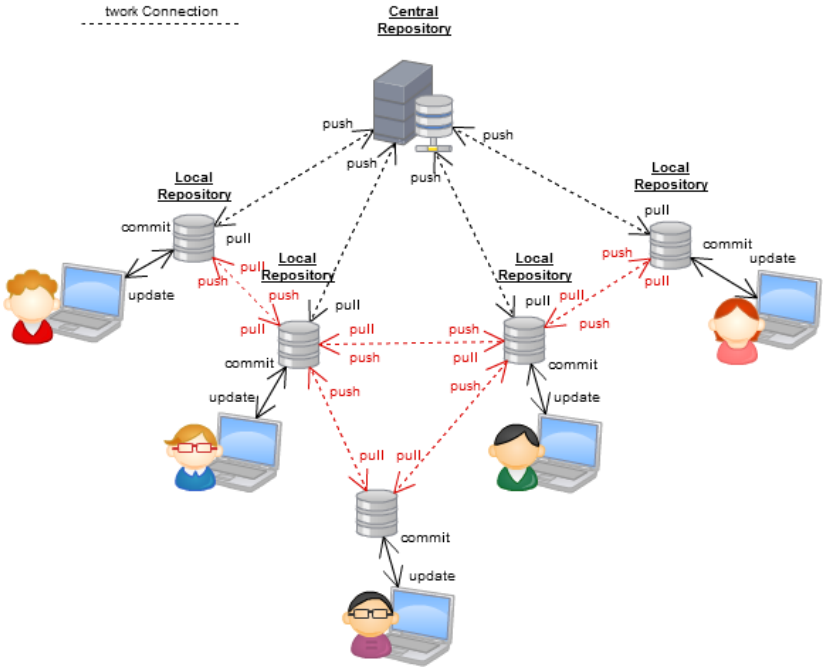
Pull

- Git pull = get fetch + get merge origin/upstreambranch
- Things can have changed in the remote repo since your last fetch
- Remember others can't see your work until it's on a remote server
- Fetch fetches the objects and updates the /refs/remotes/



What we have learned so far





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
```

- Easy way to let people contribute, without giving away control
- Policy to enforce quality even internally

Danske Bank

Exercise

Git reset

- Reflog –hard
- Git reset can change what the branch pointer points at
- Git reset can change the working tree, the staging area
- Undo last commit
- The HEAD
- Force update, loosing work

```
➢ git reset HEAD~1 #undo last commit
➢ git reflog #if you did something bad
➢ git reset #unstage
➢ git reset --hard #reset working folder
```

Danske Bank

Git rebase

- Why people do it
- Dangers / rewriting history
- How it is done.
- Rewriting your local history (interactive)

Danske Bank

Complete change of scene. Random Git stuff

By
David Stiel
dsti@danskebank.dk

Danske Bank

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.

Danske Bank

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)