Git for beginners

Klas Mellbourn
klas@mellbourn.net
github.com/mellbourn
stackoverflow.com/users/46194

Curriculum

- total time about 3h
- overview
- basics
- theory
- undoing
- ignoring stuff
- Visual Studio
- branches
- remotes
- remote branches

can be held in 3.5h if in a pinch, skip some walkthoughs then
2.5h is possible, but really too little time.

What is Git?

- *Distributed* Version Control System
- each repository independent and has complete version history
- communication at well defined points
- most commands local
  - commit
  - branch
  - merge
  - history

- fast
- offline is the normal case
- each clone is a backup

not really a client-server relationship
each repo has a complete version history

Git Clients

- Open source
- many clients
  - git command line
    - with powershell module: posh-git
  - Git Extensions (stand alone gui client)
  - TortoiseGit (stand alone gui client, shell integration)
  - SourceTree (cross platform gui client)

- Visual Studio Microsoft Git source control Provider
  - built in to all versions
  - VS2015 even has method level git history

some GUIs tries to hide what makes git special
Microsoft committed to git

also woth mentioning: sourcetree

Absolute basics

- create a file
- create a repository
- set username
- commit
- show history

- change
- show history

note that no central repo is necessary
note: when in a directory with a .git repo, the git tools detect this

Make several lines in the files

mkdir c:\course
mkdir project
add a text file foo.txt
add lines 1,2,3,4,5

create new repo
bring up git extensions
set up username

commit

make change
commit

Local

Note that we have no server (yet)
You can always quickly create a git repository

Personally I often create version controlled folders at home, without any remote

Easy and fast to creat repos. Notice how many on Bitbucket already

Staging

more changes

1. add files
2. stage one, commit it
3. stage another, commit it
4. change file
5. partially stage
6. commit
7. view history
8. note that you can see the file tree in each commit

add modifications “feature 1”, “feature 2”

Delta storage vs Directed Acyclic Graph

Not deltas,
Snapshots!

show 43-58

Content addressable

each blob and tree has a content hash as it’s name.
the tree points to the blob using the hash!

used internally by git to figure out if a commit needs to be fetched from a remote or not

Note that it is very hard in a truly distributed system to have sequential version numbers

references

show 136-152, show trees in git extensions browser, show the parent pointer
+7min=30min

Undoing things

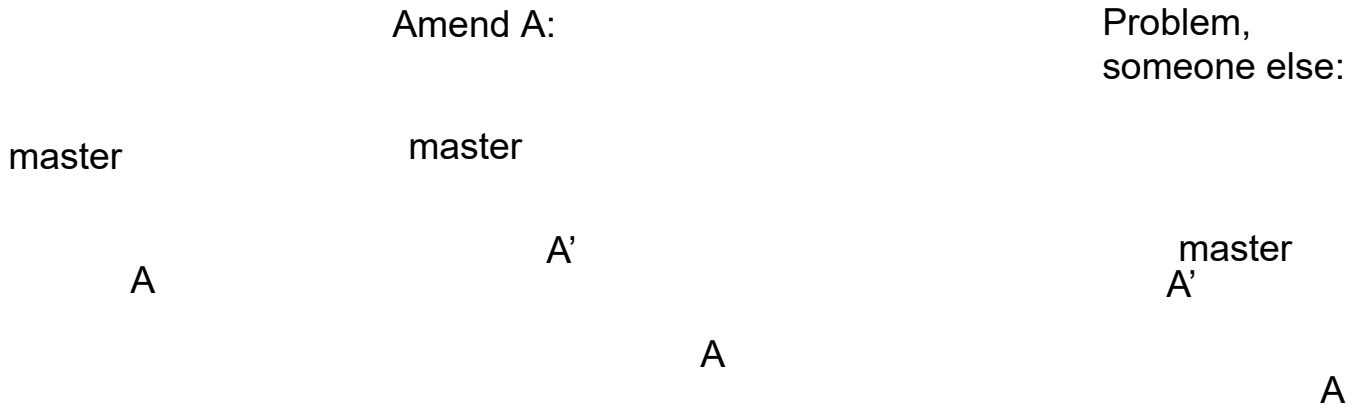
- resetting working directory
- reverting commits
- reset - move branch to a commit
  - hard
  - soft

- amending commits

reset using commit dialog

mention: revert a merge
jump back and forth using reset
you said that you cannot loose committed stuff in git
+6min

amend



.gitignore

- Some files you don’t want to version

control

- Put them in .gitignore
- Demo: Visual Studio
- note that the tree of the commit does not contain all files

First show ignoring in simple solution. foo.dll

start by using Visual Studio here

ConsoleApp_alice

ignore *.orig

(remove text auto attribute)
git config --local --unset merge.tool (since no three way merge)
(git config mergetool.prompt false)
set name
git config --local user.name “Alice”
git ocnfig --local user.email alice@hm.com

show change

Branching

In git you branch a lot!

- branch
- checkout (switch branch)
- merge
  - reintegration merge
  - conflict resolution

feature: math and add output
master: rename Program and add output

Note: in git you do not create a new folder for a new branch

fishbone pattern

+26min. This is how far I actually got in two hours 2014-06-03

Remotes

- git clone
- remotes
- remote branches vs local branches
- push
- pull

point out that you need only files: no registry, no database, no server processes
create two repositories Alice and Bob
do pushes and pulls

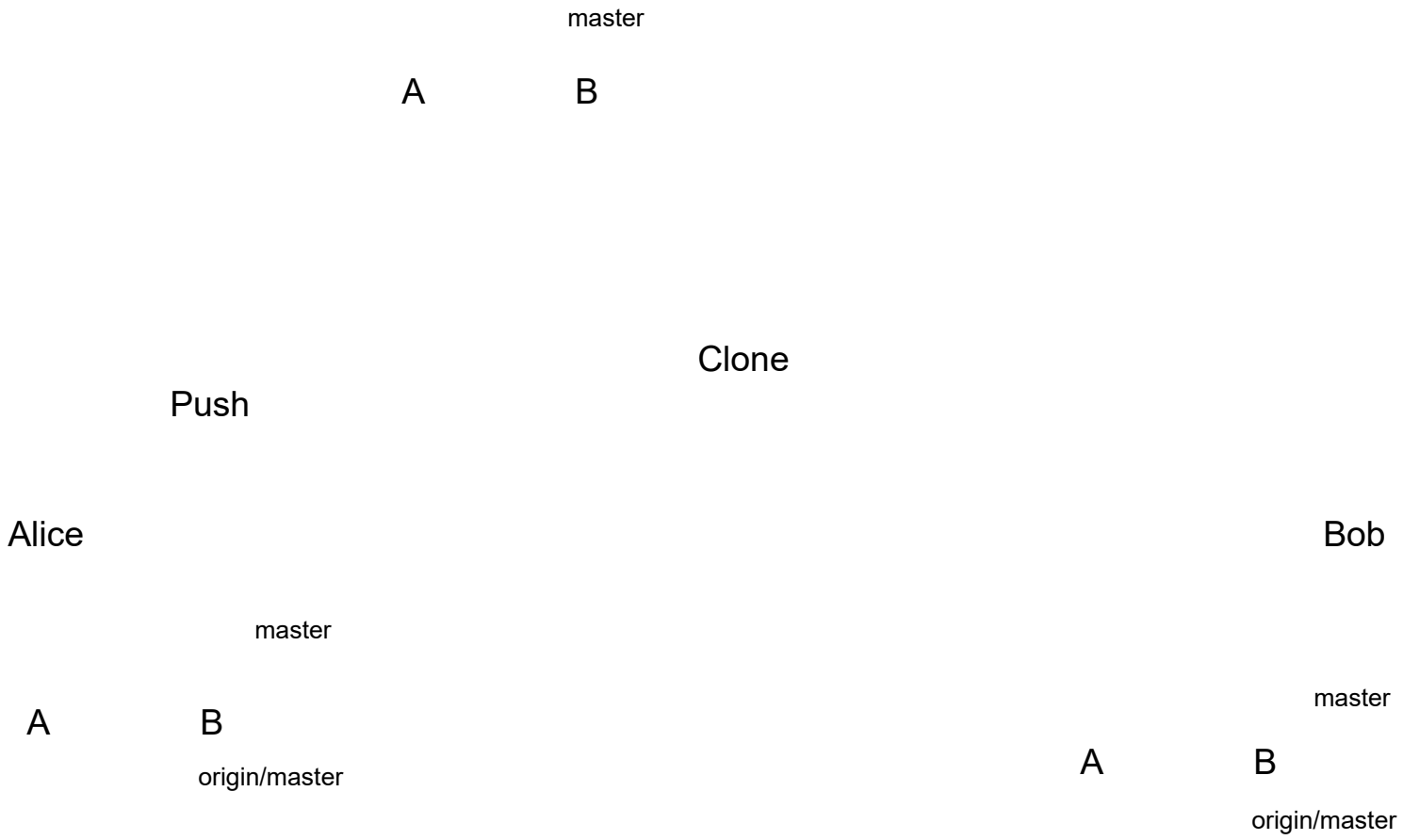
clone Alice to a central repo or push -u

do all using Git Extensions!
(git remote add orgin ..\ConsoleApp.git
git fetch
git branch -u origin/master)

git clone ConsoleApp.git bob_ConsoleApp
git config user.name Bob
git config user.email bob@hm.com)

Clone

central repo - “origin”remote



push/pull - the simple case

central repo - “origin”remote

master

A B C

Pull

Push

Alice

Bob

master

A B C

origin/master

master

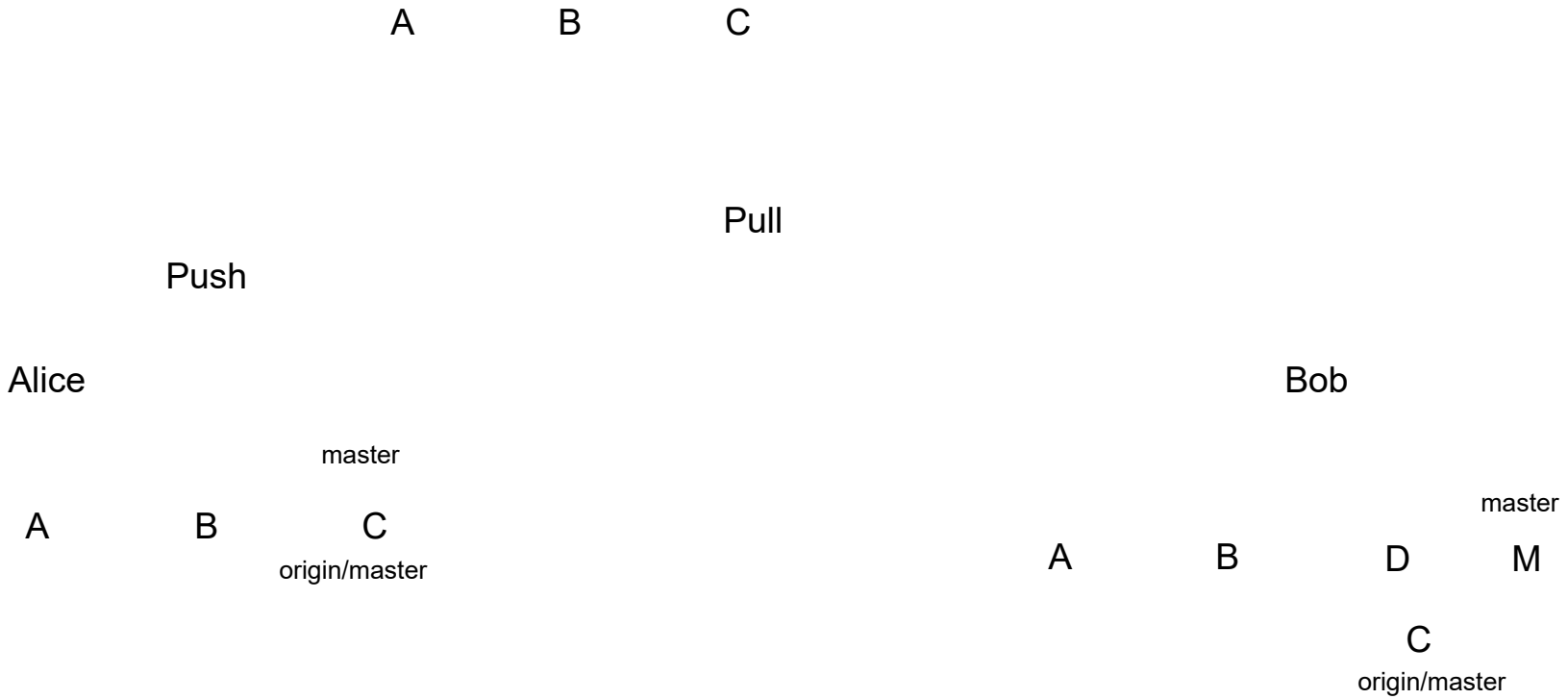
A B C

origin/master

Push/pull with merging

central repo - “origin”remote

master



show rejection
pull = fetch + merge

example changes:
Math class
renaming stuf

feature branch workflow

1. create a feature branch
2. checkout feature branch
3. make commits until done with feature
4. checkout master
5. pull master
6. merge feature into master
7. make sure everything works
8. push master

feature branch example

you should not normally do development on master, branch!
isolated environment, clearer history

update your feature branch

- Merge from master to integrate latest changes
- fishbone pattern:

git handles reintegration merges well,
takes advantage of earlier conflict resolution

remote branches

- You can push and pull other branches than master
- This way you can share work on a feature branch
- Branches that are local on the remote are reproduced as “remote branches” in your repository
- Remote branches cannot themselves be checked out
- Local branches can *track* remote branches

remote branches are in the local repository,

create feature branch that they should collaborate on (multiplication/division)
explain colours in git extensions

let alice make a bug correction in master and push it

mention teamcity build possibility
+15min

Tags

- label a version with a tag
- pointer to a commit
- if annotated it also has
  - comment, tagger, date
  - hash
- does not move (like branches do)
- cannot be checked out (like branches)

- must be pushed specifically

wait, you have been nagging us about how a branch is a pointer to a commit, WTF?
reference tag vs annotated tag

stash

- Save changes “temporarily”
- clears any changes from working tree and staging area
- can be applied later
  - even on different commit

- can popped or applied or dropped

show first one stash, pop it,
save it again using a name
save another stash
demonstrate the stash list

Rebase

- A rebase replays the differences introduced by

commits, creating new "copied" commits.

- cleaner history
- fewer commits
- don't do it with pushed commits

sign Git to Manage..." slide 218-227
fewer
show how to make a feature branch with no commits
pull with rebase

use same branch as before, just delete the remote

I just barely finished this in another 2h (total 4h with two 10min breaks), skipping briefly over tags and stash
another 30min was good for more about tags, stash and more rebase

Links

gitref.org - quick walkthrough
git-scm.com/book - compete book, especially Chapter 2 (and preferably also chapter 3)
youtube.com/watch?v=ZDR433b0HJY - excellent video on basics (but very command line oriented)
youtube.com/watch?v=ig5E8CcdM9g - advanced video
How to use Git on GitHub

some of the graphs in this presentation are from these links

2h52min with ten minute break

More goodies:
Bitbucket/github

fork/pull request
Dynamic branches on TeamCity
3h45min inclusive of goodies