

# Revision Control Systems ...mostly Git

---

Spring 2016  
Alfred Bratterud



# Agenda:

- \* Revision control: Why, What and which
- \* Distributed vs. Centralized RCS.
- \* Git A-B-C
- \* git servers
- \* Exercises



# What is a revision control system?

«Revision control, also known as version control and source control (and an aspect of software configuration management), is **the management of changes to documents**, computer programs, large web sites, and other collections of information.»

[http://en.wikipedia.org/wiki/Revision\\_control](http://en.wikipedia.org/wiki/Revision_control)



# It's really important

## **The Joel Test**

1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?

From <http://www.joelonsoftware.com/articles/fog0000000043.html> 11-01-2015



# It's really important

## The Joel Test

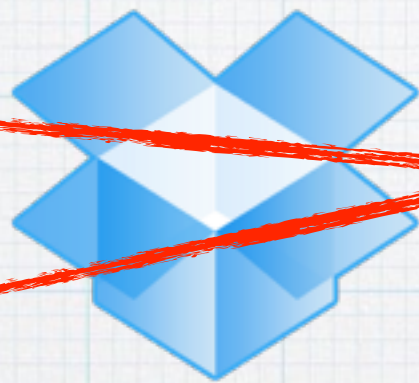
1. Do you use source control?
2. Can you make a build in one step?
3. Do you make daily builds?
4. Do you have a bug database?
5. Do you fix bugs before writing new code?
6. Do you have an up-to-date schedule?
7. Do you have a spec?
8. Do programmers have quiet working conditions?
9. Do you use the best tools money can buy?
10. Do you have testers?
11. Do new candidates write code during their interview?
12. Do you do hallway usability testing?





**Dropbox**





**Dropbox**

...It's better than nothing but you don't get  
any points in the Joel test



# So what *\*exactly\** is revision control?

- \* Every change results in a new, unique «state» or «version», with a unique identifier (which is forever).
- \* Every change to a document is tracked - forever (by default).
- \* Every change is associated with a timestamp and an author.



# Why so important?

- \* Every change results in a new, unique «state» or «version», with a unique identifier (which is forever).
- \* Every change to a document is tracked - forever (by default).
- \* Every change is associated with a timestamp and an author.



# Why so important?

- \* Every change results in a new, unique «state» or «version», with a unique identifier (which is forever). **We can roll back to location X across deployments - or know if they need a patch**
- \* Every change to a document is tracked - forever (by default).
- \* Every change is associated with a timestamp and an author.




# Why so important?

- \* Every change results in a new, unique «state» or «version», with a unique identifier (which is forever). **We can roll back to location X across deployments - or know if they need a patch**
- \* Every change to a document is tracked - forever (by default). **We can trace bugs and progress. ...this worked at location N not N+1**
- \* Every change is associated with a timestamp and an author.



# Why so important?

- \* Every change results in a new, unique «state» or «version», with a unique identifier (which is forever). **We can roll back to location X across deployments - or know if they need a patch**
- \* Every change to a document is tracked - forever (by default). **We can trace bugs and progress. ...this worked at location N not N+1**
- \* Every change is associated with a timestamp and an author. **So we know who to** 

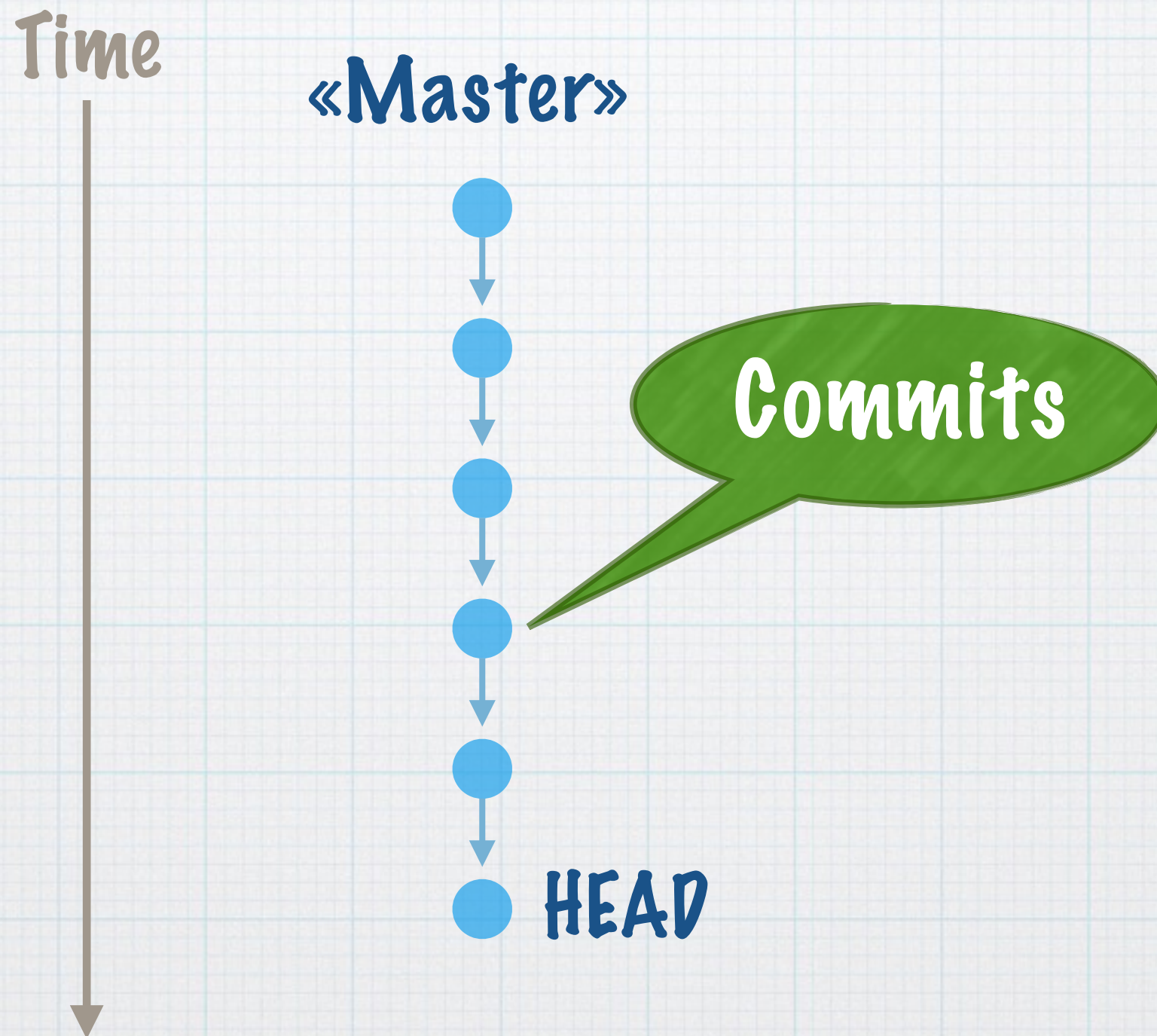


# It's a graph - and a tree

- \* So far: sounds like the perfect backup system (we'd need only 1 tape)
- \* But unlike backups, RCS's have branches
  - \* Say we have two possible features - A and B. Let's try both!
  - \* Branch swapping: We might go for A now, but maybe later, we'll want to switch to B
  - \* Branch merging: ...or we might decide to use both
  - \* We want to commit often, in branch A and B
    - \* While making sure each commit gets a unique ID

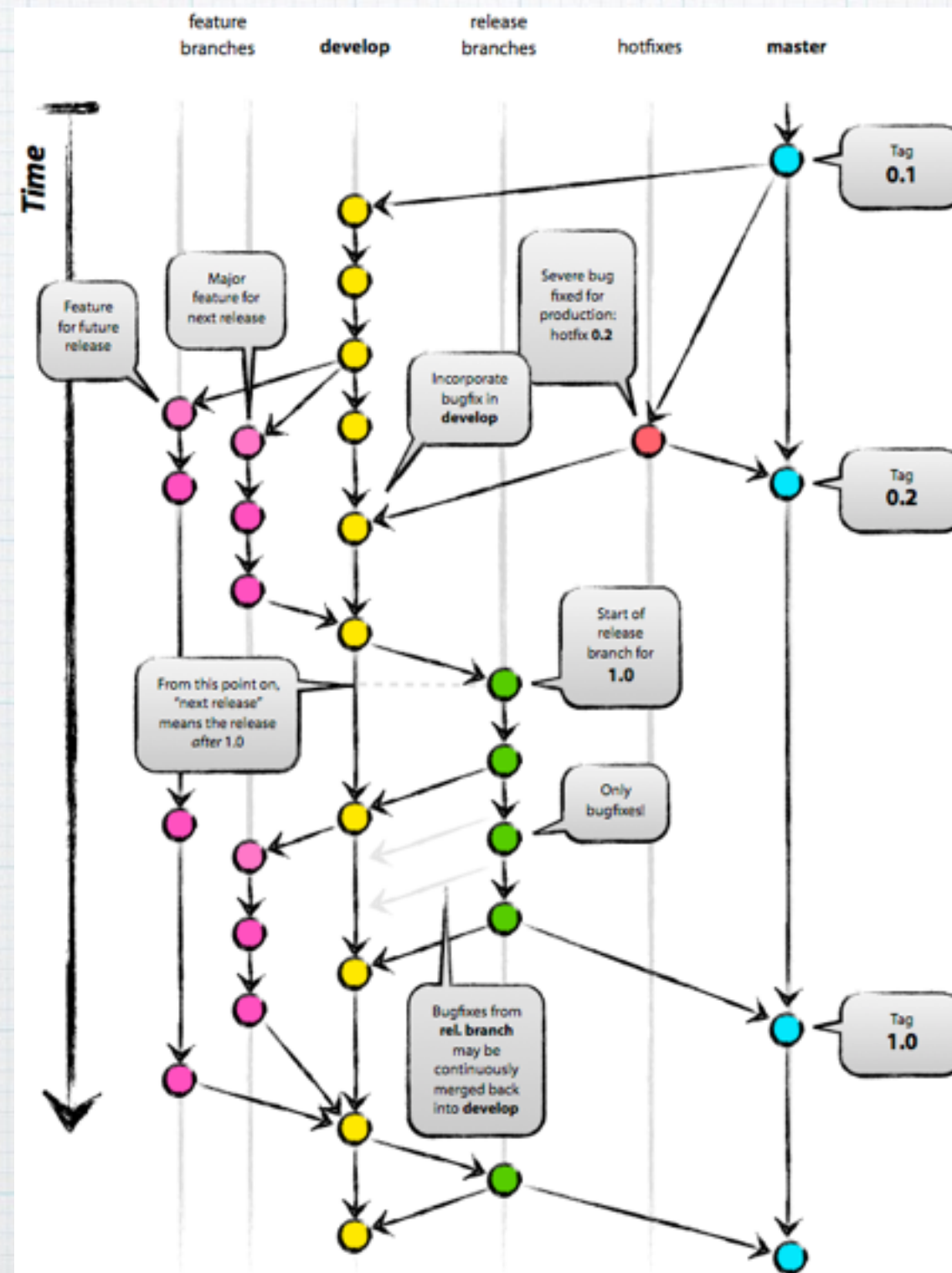


# ...Simple by default





# ...Can be complex





# It's a graph - and a tree

- \* (Good) Revision control systems are designed for an *\*active\** workflow
  - \* Creating, merging and swapping between branches
  - \* I.e. actively Traversing and modifying the source tree
- \* Unlike backups which are passive.
- \* So - A LOT of states: too much data?
  - \* We only need to keep the diff's
  - \* Usually we handle text-only, which compresses well



# Centralized v.s. Distributed

---

Two paradigms of Revision Control Systems



# Centralized RCS

- \* The «original»; CVS, SVN etc.
- \* Everyone commits to a server - «push»
- \* ...If you have «commit rights»
- \* «The boss» decides who gets to commit (write access)
- \* Pros / Cons:
  - \* Strict control over what the «Master» copy is
  - \* Strict control over committers
  - \* You need a server
    - \* Every commit requires a server connection



# Distributed RCS

- \* BitKeeper, Bazaar, Mercurial and **Git**
- \* «No pushing» only pulling
- \* So you only need read-access to everybody else - and you can pull their changes into your «working copy»
- \* Pros / Cons:
  - \* No «service» or server required by default - just a binary. But anyone can set up a «push» server / service if they want
  - \* No politics around «push rights» - nobody pushes. If you do - it's just to move the data to a server
  - \* No «Golden master copy» by default.
    - \* Linus Torvalds has the most popular «working copy» of Linux
    - \* ...But his github-copy is the undisputed «master» anyway



# Git

- \* Distributed revision control system
- \* Linux kernel used BitKeeper; proprietary but free, until 2005
- \* Torvalds couldn't find a replacement good enough, and decided to make his own.
- \* GitHub uses git directly as it's back-end
  - \* In 2011 it was reported that Github had surpassed both SourceForge and Google Code in popularity \*
- \* In 2014 Eclipse reports that git has surpassed subversion (SVN) as the most deployed revision control system in the world \*\*

\* <https://ianskerrett.wordpress.com/2014/06/23/eclipse-community-survey-2014-results/>

\*\* <http://readwrite.com/2011/06/02/github-has-passed-sourceforge>



# Linus on Git



#X!%?!&!

<https://www.youtube.com/watch?v=4XpnKHJAok8>



# Git A-B-C

---

Just enough to be extremely useful



# Setting up git on Ubuntu

- \* `$ sudo apt-get install git`  
It's just a binary - so no noise
- \* `$ git help / $ man git`
- \* `$ git init`  
Makes the working directory into a «git repository»
  - \* It just adds a «.git» folder - nothing else
  - \* All past and future versions will be kept here (but unreadable)
- \* That's it!
  - \* You now have a state-of-the-art revision control system
  - \* And a «repository» set up to receive changes



# Basic Git commands

- \* Whenever you add a file or folder - you have to add it with `git add .` or `git add my_file.txt`
- \* `$ git status` : tells you what has happened since your last commit
  - \* Use it often! It will also tell you about untracked files
- \* `$ git log` : shows you the git messages from previous commits. If messages are used properly, this gives you a great summary of the progress



# Basic Git commands

- \* `$ git add <pattern>` : Adds whatever matches to «version control». (Just use «.» as pattern for «everything here»)
  - \* i.e. from now on - keep track of these files / folders
  - \* But - this does NOT add future files to version control
  - \* Nothing actually gets tracked without further commands (no service)
- \* ...Do some changes - try to do «1 thing» pr. commit
- \* `$ git commit -am «Fixed this 1 thing»` : Scan anything added with git add, create a new unique «commit» containing the diff between now and the last commit.
  - \* - a : Commit all changes
  - \* - m: A commit message (Compulsory - you have to say what you did)
- \* Do more changes, and commit again. Repeat.



...Let's try

---



# git / Revision Control for sysadmins?

- \* Do sysadmins need text documents?
  - \* Then yes!
- \* Puppet classes
  - \* Clients can automatically pull the latest revisions from a central git repository
  - \* Git will detect if they're up-to-date
    - \* if yes, it fetches incrementally
    - \* if no, it hangs up
- \* Ideal for scripting - that's just programming
- \* Documentation? LaTeX? Iptables rules? Sure!!
- \* For DevOps: Even more so. More on that later.



# Git on the server

---

...We like distributed - but why not centralized too?



# ...Just add ssh

---

Really!  
(Well, kind of)



# Centralized git server in 5 minutes

- \* In general, pulling is better than pushing
  - \* Only requires read-access every way
- \* A simple git-repo of puppet classes: Any server with git and ssh - enough to scp from
  - \* Create a linux user with read-access to a git-repository (like the folder we just created)
  - \* Add ssh security as needed (i.e. add keys)
  - \* On a client (git over ssh syntax just like scp):  
`git clone puppet@puppetmaster.mydomain.local:folder_name`
  - \* ...Later you do some changes on the server
    - \* Commit them using `git commit`
  - \* Clients can now fetch changes with `git pull`:
    - \* `git pull origin master`
    - \* «origin» is just a name, referring to the (ssh) link
    - \* «master» is the name of the branch you want to pull to



...Let's try

---



# But, some times pushing is nice...

- \* We might not want everybody to run ssh-daemons on their laptops - but we still want to pull their code
- \* For a centralized git-server, we might not want to work locally on the server all the time - or log into it and pull from somewhere else
- \* Git allows for pushing - but only to «bare» repositories
  - \* They only have the «.git» part, and no working directory
  - \* I.e. Repo's that you can't work on locally, but only push to / pull from
- \* Github repos are «bare» - they are folders with «.git» extensions, like «repo\_name.git»
- \* It's an exercise to set this up. Once done - push with:
  - \* `git push origin master`



# Github / gitlab

- \* Essentially: Web GUI layers on top of git
  - \* Very nice source tree navigation
  - \* Statistics / graphs etc.
  - \* Built-in issue tracker
    - \* Issue numbers used in git messages automatically links to issue
- \* Github is proprietary, but free to use for open source projects
  - \* Private repo's cost money
  - \* Students get 5
  - \* Private installation costs money (github enterprise)
- \* Gitlab is dual license
  - \* Free Open source version, can be deployed privately
  - \* We have one!



# Git branches

- \* Show branches with  
`git branch`
- \* Create a new branch with  
`git branch my_branch`
- \* A new branch is identical to whichever branch you were in when you made it
- \* Switch to a branch with  
`git checkout my_branch`
  - \* Do changes, commit as usual
  - \* Switch back: The whole file tree is automatically reverted



# A new branch

Time

«Master»



HEAD

git branch my\_branch



# A new branch

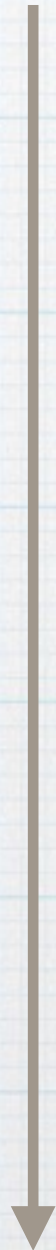
Time

«Master»

HEAD

git branch my\_branch

checkout + commit





# A new branch

Time

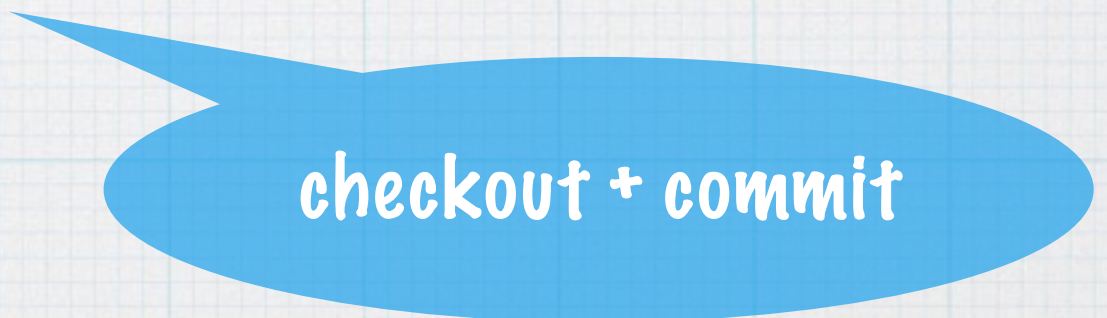
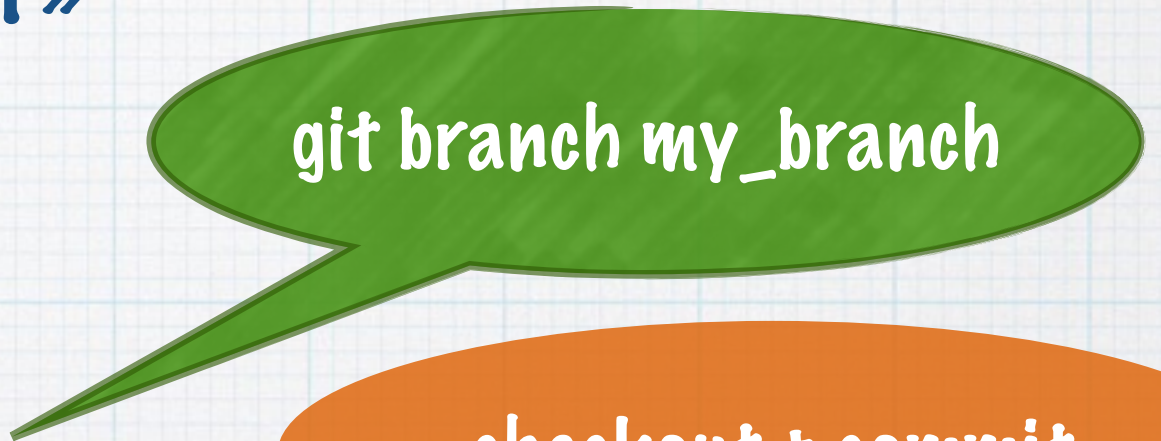
«Master»

HEAD

git branch my\_branch

checkout + commit

checkout + commit





...Let's try

---



# Moving and removing

- \* Assume you're inside a git repository and you move a file like so:
  - \* `mv policy.txt policies/policy1.txt`
  - \* Git now sees «policy.txt» as deleted and policies/policy1.txt as a new file.  
(It would be crazy to check every new file against every file in the tree to see if they were identical)
  - \* Problem?
    - \* The git history for policy.txt is not linked to policies/policy1.txt - they are totally unrelated
    - \* If «Old Boss» did a lot of work on the policy, nobody will know, unless they dig way back
  - \* Solution: `git mv` - will do the move both in the directory tree, and in the version control tree.
- \* Similarly - remove with `git rm`. Otherwise the file will be removed from the file system, but not from version control. The next pull, you might get it back.



# .gitignore

- \* You'll usually want to prevent git from tracking certain things - like `my_script.sh~`
- \* In any repository, you can add a `.gitignore` file with a list of shell-patterns to ignore
  - \* One line pr. pattern - wildcards only - no regexes
- \* You can add the `.gitignore` file to version control if you want
  - \* Like anything hidden with «`.file_or_folder`»
  - \* Note: Git only tracks content - empty files / folders are ignored
- \* Examples:
  - \* `~`
  - \* `my_binary`
  - \* `\#*`
- \* All the details: `man gitignore` / `git help gitignore`



# Key concepts:

- \* Source control is not backup
  - \* Each change is saved, tagged and ID'd «forever»
  - \* Git supports active use of branch merge / swap
- \* Distributed v.s. Centralized
  - \* Distributed pretty much rocks
  - \* Corresponds to Pull vs. Push
- \* The «git index» / source tree is separate from the file tree
  - \* Git tree consists of historical versions of the current (git controlled) file tree
  - \* If something is in your «working tree» (file tree) does not mean it's in the «git index» / source tree.



# github demo / Exercises

---

ALL things GIT:  
<http://git-scm.com/doc>