GIT

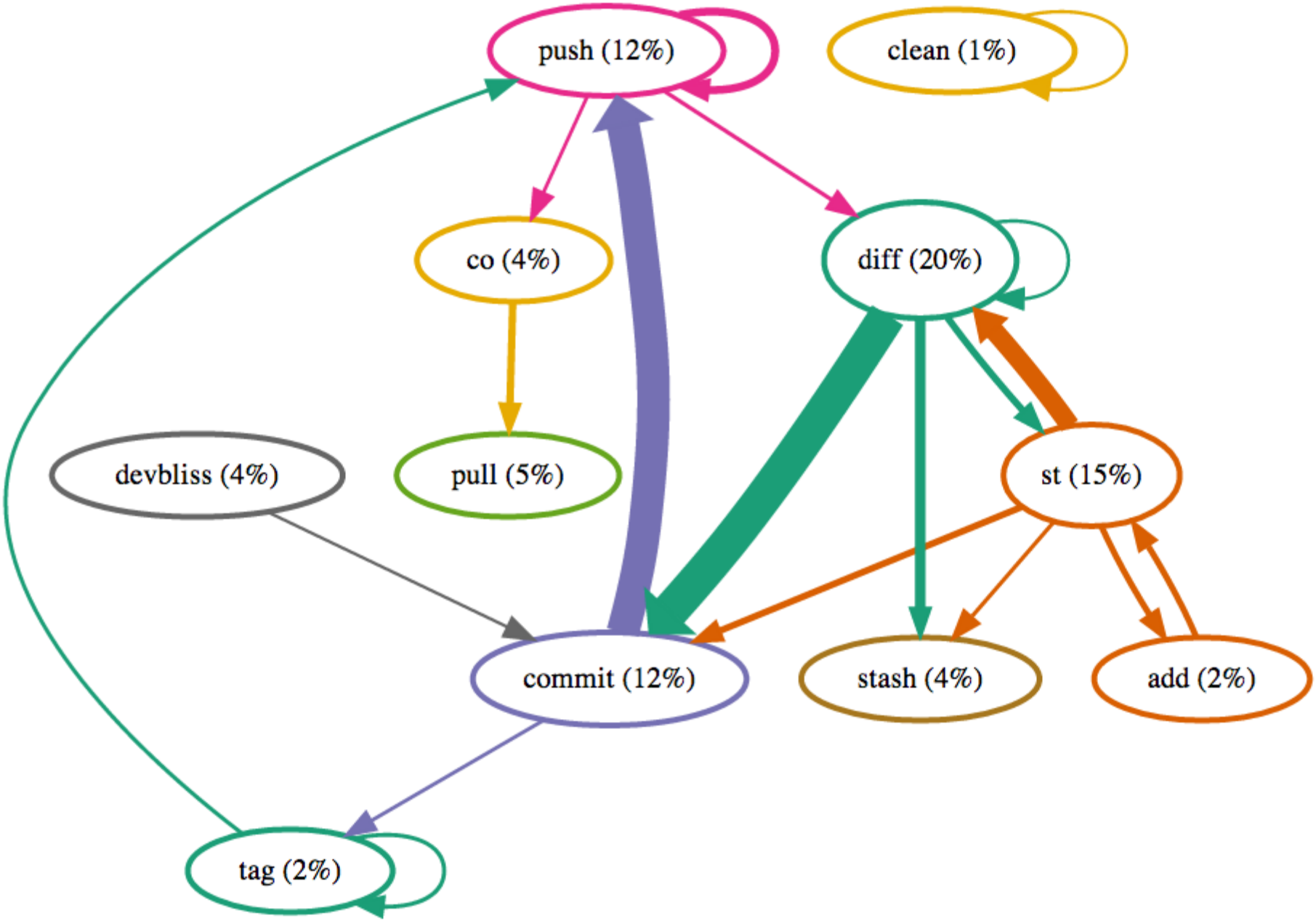
Porcelain

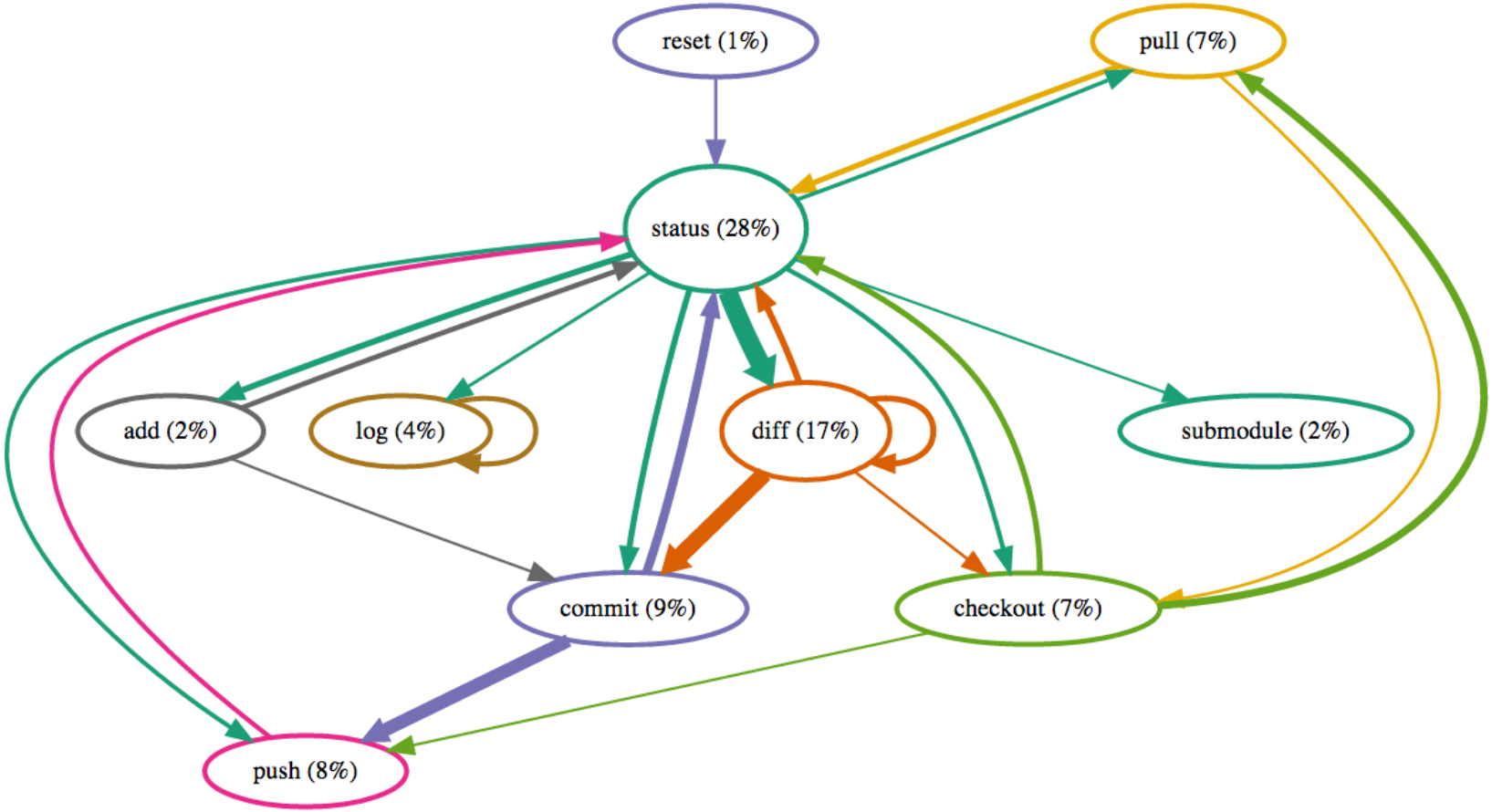
From code to commit

- git clone git@github.com:devbliss/scripts
- git checkout -b feature/new_script
- vi cool_script # code
- git add cool_script
- git commit -m ‘added cool_script’
- git push

Example Workflows

https://visualize-your-git.herokuapp.com/


push (12%)
clean (1%)
co (4%)
diff (20%)
devbliss (4%)
pull (5%)
st (15%)
commit (12%)
stash (4%)
add (2%)
tag (2%)


reset (1%)
pull (7%)
status (28%)
add (2%)
log (4%)
diff (17%)
submodule (2%)
commit (9%)
checkout (7%)
push (8%)

Types of commands

- Gather intel (status, diff, log, blame)
- Transfer between areas (add, reset, commit)
- Modify history / code (reset, rebase, revert)
- Modify references (branch, tag)
- Talking with remotes (remote, push, pull)
- Manage submodules (submodule)
- Manage the repository (clean, gc, reflog)
- Plumbing

Basics

<tree-ish>

- HEAD
- master
- origin/master
- master@{1} # previous reflog entry
- master@{2.minutes.ago} # reflog entry
- HEAD^n   # n-th parent
- HEAD~n   # HEAD^1 times n
- master@{1.days.ago}~4
- HEAD~4^2~3
- HEAD~~~^2~~

<range>

- What's in master that is NOT in origin/master
  - git log origin/master..master
  - git log ^origin/master master
  - git log master --not origin/master
- What has changed in feature and master since branching off?
  - git log master...feature/coolfeature
  - git log --left-right master...feature/coolfeature

Config

- color.diff=auto
- color.status=auto
- color.branch=auto
- merge.conflictstyle=diff3
  - show mine, theirs, parent instead of
- push.default=current
  - or simple. default is matching (push ALL branches)
- alias.de=devbliss
- set user.name, user.email (github, hooks)

Gather Intel

git status

- Get current branch
- Untracked files
- (Un)staged changes
  - new file
  - deleted
  - modified: bin/scripts (new commits)
- Conflicts
- # of commits ahead / behind origin

git diff

- git diff --staged # what is staged?
- git diff master@{1} # what did I just pull
- git diff master..origin/master # after fetch!
- git diff --color-words # word-diff
- git diff --color-words=. # char-diff
- git diff -w # ignore whitespace github: ?w=1
- git diff master <filename>

As always: Combine!

git diff --color-words=. -w master@{1} Makefile

git log

- git log -p # patch => show diff of commit
  - git log -p -w # ignore whitespace (just like diff)
- git log --decorate # tags, branches
- git log --graph # draw tree graph
- git log --stat # show +/- for each file
- git log -n3 # show only first three log entries
- git log <filename>

Can also be used for commit-based diff!

- git log --left-right master...feature/branch

git blame

- git blame Vagrantfile
  - what is the oldest revision that already had that line
- git blame --reverse <start>..<end>
  - git blame --reverse HEAD~20..HEAD
  - what is the highest revision that the lines from <start> are still in?
- git blame -M
  - detect moved lines
- git blame -C[-C[-C]]
  - detect lines copied from other files in same commit
  - “ in same commit and in creating commit
  - “ in all commits

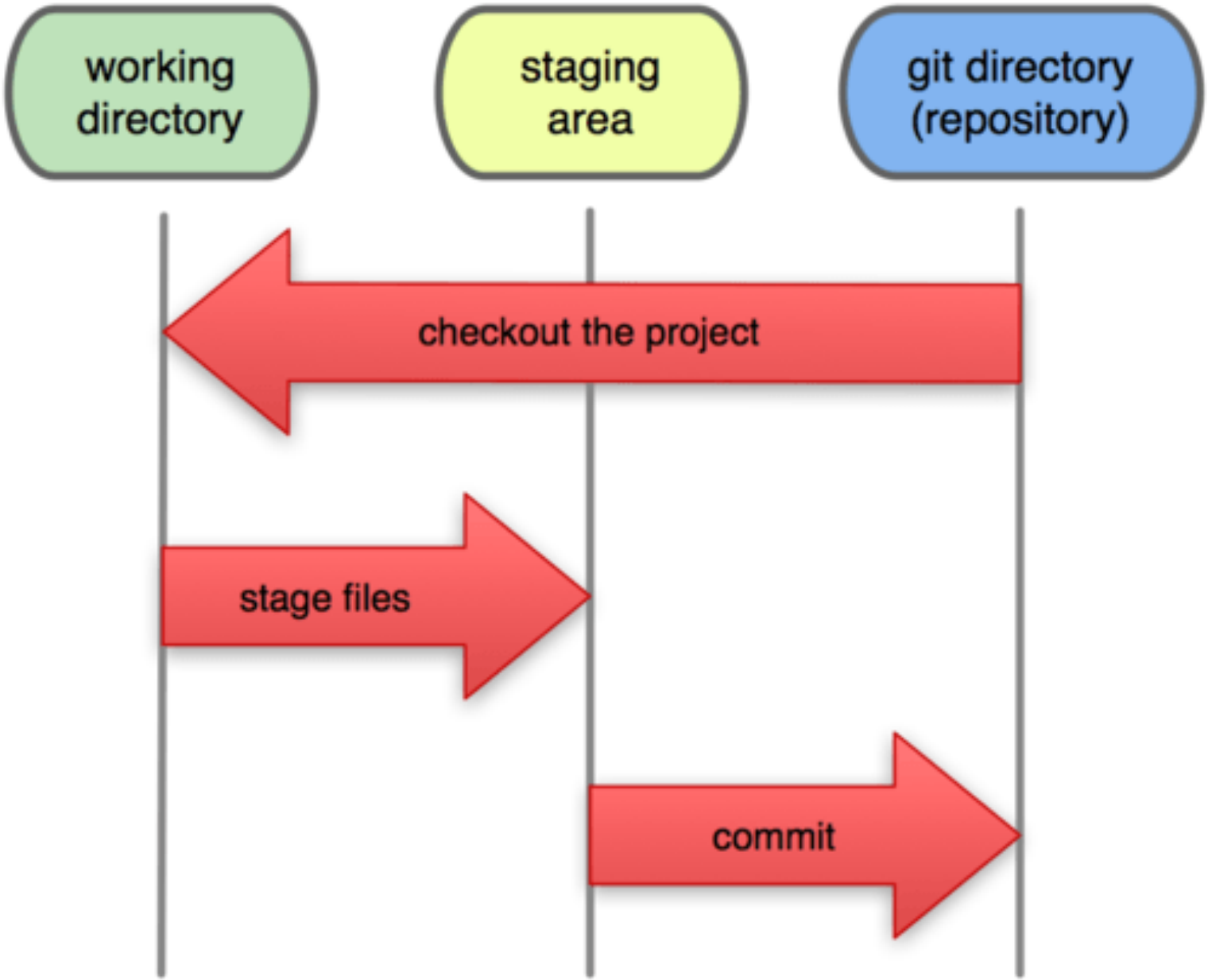
git bisect

- binary search all commits for the first bad
- if HEAD is defunct and tag 1.0.0 is good:
  - git bisect start
  - git bisect bad # no hash /ref => HEAD
  - git bisect good 1.0.0
  - git will checkout a commit between good and bad
  - run tests
  - git bisect good / git bisect bad
  - repeat
- git bisect reset # to stop bisect mode
- git bisect run   # auto bisect!

Transfer between areas

Local Operations

LOCAL!!!



How to (un)stage changes

- git commit -a # stage all CHANGED files
  - potentially disruptive
  - only use after git status
- git add . # stage changed and new files
- git rm <file> # stage deletion of file
- git add -A # stage changed, new and deleted
- git reset <file> # unstage file

How to (un)stage interactively

- Interactive add
- Can (a)dd untracked, (u)pdate = stage, (r)evert = unstage, (p)atch update
- good for fast selective patch updates

git stash

- git stash # stash uncommited
- git stash apply # apply most recent stash
- git stash list [-p] # show stashes [with diffs]
- git stash show [-p] stash@{n}
  - for specific stash

Useful for before pulling if not ready to commit

Modify History / Code

git reset - change history

- git reset --hard origin/master # undo a merge
- git reset HEAD~1
  - last commit is “undone”
  - content is “not staged for commit”
- git reset --soft HEAD~1
  - last commit is “undone”
  - content is staged.
- git reset --hard HEAD~1
  - force your HEAD to HEAD~1
  - the old HEAD is still out there, but without reference

git commit

- git commit --amend
  - Basically this combines two commands:
    - git reset --soft HEAD~1
    - git commit
  - Useful if you forgot to include something in the last commit, or if you want to change the commit message AND DID NOT PUSH YET

git checkout - more than switch

- git checkout master # switch to branch
- git checkout master <file> # get from branch
- git checkout --ours <file> # resolve conflict
- git checkout --theirs <file> # resolve conflict
- git checkout <file> # reset changes
- git checkout -p <file> # patch reset

git revert

- git revert <hash>
- creates a commit with negative diff to cancel out the reverted commit
- the old commit stays!
  - important for reverting a merge

git rebase

- Short:
  - Don’t
- Long:
  - Merge commits are really useful
    - can be reverted
    - time and person who merged is recorded
    - default of github pullrequests
  - rewriting history can be really ugly
    - merge of rewritten + original => duplicates
    - information is lost
  - Don’t :p

Modify references

git tag

- git tag 1.0.0 # create a lightweight tag
- git tag -a 1.0.0 # create a real tag
- git tag -s 1.0.0 # create a signed tag
- git tag -f 1.0.0 # update tag (local!)
- git tag -d 1.0.0 # delete tag (local!)

- Never delete a remote tag to change it!
  - A tag is only ever downloaded once. If it changes remote, it does not change locally

git branch

- git checkout -b <branch_name>
  - git branch <branch_name>
  - git checkout <branch_name>
- git branch --no-merged master
  - show all local branches not merged into master
- git branch -r --merged origin/master
  - show all remote branches merged into origin/master
  - those can be deleted :-)
- git push origin :<branch_name>
  - delete a branch

Talking with remotes

git remote

- git remote set-url origin <new_origin_url>
  - if you move the repository
- git remote add upstream <upstream_url>
  - get upstream changes into your fork
- git remote -v
  - show all upstreams with urls
- git remote prune origin
  - delete all remote tracking branches that are deleted on the remote

git fetch / push / pull

- git fetch
  - gets all branches, all tags
  - local copy of all saved in .git/refs/remotes
- git pull
  - git fetch && git merge origin/<tracked_branch>
- git push
  - 1.x default: push ALL matching remotes to server
  - 2.x default: push current matching branch to server

Manage submodules

git submodule - Setup

- Very easy to add a submodule:
  - git submodule add git@github.com:user/repo
  - git commit -am ‘added submodule’
  - git push

- But what did actually happen?

git submodule - where is the URL?

- submodule URL is under git version control:
  - <project_root>/.gitmodules
    [submodule "<submodule_path>"]
    url = <submodule_url>
- There is a local copy in .git/config that is used for the actual commands!

git submodule - where is the hash?

- submodule commits are saved directly within each commit:
  - git ls-tree HEAD # for submodules in project root
  - git ls-tree HEAD:<folder_with_submodules>
- Every commit can be checked out at any time later, and the submodules are compatible (as they were when that commit was pinned)

git submodule - get it step by step

- git pull:
  - get .gitmodules (under version control!)
  - get HEAD (which includes submodule commits)
- git submodule init
  - copy entries from .gitmodules to .git/config
  - OR: .git/modules/<path-to-submodule>/config
- git submodule update
  - clone/update submodules to pinned commit

git submodule - get it faster

- git submodule update --init
  - combines init and update => no local modification
- git submodule update --init --recursive
  - as above, but does this recursively for all submodules
- git clone --recursive
  - Only on first clone, but will clone all submodules along

git submodule - status

- git submodule [status [--recursive]]
  - hash for every submodule
  - + => hash does not match with upstream
  - - => not initialized
- git submodule summary
  - commit headlines for changes between current and remote revision
  - only for direct submodules, no recusive!

git submodule - foreach

- git submodule foreach [--recursive] <cmd>
  - runs command on all submodules
  - stops on first exitcode > 0

Appliances:

- git submodule summary # for all submodules
- git pull origin master # update all submodules
  - This is only ok for bleeding edge development!

git submodule - change

- change pinned version
  - ‘git add’ the folder, commit and push.
  - all others have to pull and ‘git submodule update’
  - git commit -am can mess things up!
- change upstream URL
  - edit .gitmodules
  - git submodule sync && git submodule update
  - tell everybody to pull + sync + update.
- sync vs init:
  - init only creates an entry (will not update)
  - sync only updates an entry (wil not create)

git submodule - remove

- git submodule delete? - No!
  - git submodule deinit => deletes entry in local .git/config (or do it yourself)
  - git rm <path-to-submodule> => deletes folder
  - Entry in .gitmodules has to be deleted manually

Secret / Binary in Repo - Why bad?

Why is this bad?

- Binary:
  - Every time a binary changes, copy in history
  - history has all versions of all binaries
  - git get’s real slow real fast (jenkins, local)

- Secret & Binary:
  - Deleting is useless (still in history!)

Secret / Binary in Repo - Fix it!

- git filter-branch --prune-empty --index-filter ‘git rm -rf --cached --ignore-unmatch <file. name>’ --tag-name-filter cat -- --all
  - -- --all: work this command on all revisions
  - --index-filter: run filter on every commit’s index
    - git rm -rf --cached --ignore-unmatch <filename>: delete the file in the index, no error if not found
  - --prune-empty: delete commits that are now empty
  - --tag-name-filter: run filter on every tag
    - cat: use same name => update tag - DANGER!

Secret / Binary in Repo - Gotchas

- Tell everyone to merge all branches
- No new branches anymore!
- Pull master
- git filter-branch --prune-empty --index-filter ‘git rm -rf --cached --ignore-unmatch <file. name>’ --tag-name-filter cat -- --all
- git push --force (!)
- git reset --hard origin/master for everyone

Secret / Binary in Repo - Avoid it!

- if anybody forgets to reset --hard and instead does a git pull:
  - all old comits are still there
  - all the non-binary commits are duplicated

=> DO NOT PUSH BINARIES TO REPO!

Accidental Merge - Revert Merge

- on master
  - git log --graph
  - * commit <merge-sha>

    |\ Merge: cadc526 835716e
  - Check which of the commits is the good one (should be the left). left is 1, right is 2.
  - git revert -m 1 <merge-sha> && git push

Accidental Merge - Revert Merge

- on feature-branch:
  - git merge master
  - git revert HEAD && git push # revert the revert
  - Do not forget to revert the revert! (else feature defunct)

Only possible if there is a merge commit!

github does ‘git merge --no-ff’ for pull-requests

Accidental Fastforward Merge

If there is no merge commit:

- reset master:
  - git reset --hard <last-non-bad> && git push --force
  - git cherry-pick <all-other-good>
  - git fetch && git reset --hard origin/master # all others
- revert and revert-revert
  - git revert <all-bad> # on master
  - git revert <all-revert-commits> # on feature-branch