An Introduction to Git Talk

Ehsan Zandi

Deutsche Telekom IT

ehsan.zandi@telekom.de

23.08.2021

Overview

- Git vs SVN
- Git Basics
- Stash Area
- Branches
- Remote Repositories
- Undoing Changes
- Reflog
- Rebase
- Tag
- Git Objects
- Git SVN
- To be learned

Git vs SVN

- Git is a fully distributed version control system (VCS)
- Each user (PC/Laptop) is an exact clone of the remote repository
  - Each user is a repository (log, revert, merge, branch, etc)
  - No network connection required, except to sync with central repo (pull/push/fetch)
  - merge and rebasing can be done offline
- Git is much faster than SVN
- Git's repositories are much smaller than SVN
- Git's branches are much simpler and less resource heavy than SVN
- Git is much better in branch auditing and merge handling
- As many backups as the number of users
- Content integrity using SHA-1 hash

Git vs SVN

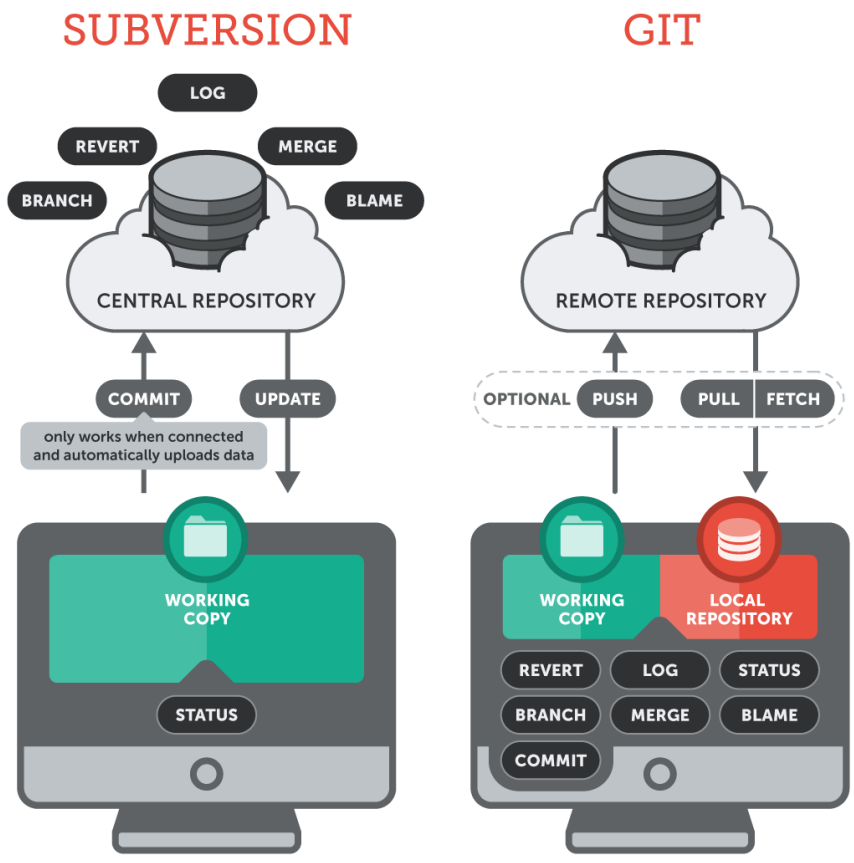

Figure: Centralized vs distributed VCS (Source: www.git-tower.com)

Git vs SVN

	SVN	Git
License	Open-source (Apache)	GNU
Distributed-ness	Centralized	Fully Distributed
Speed	✗	✓
Storage	✗	✓
Integrity Guarantee	✗	✓
Brnaching & merging	✗	✓
Stashing	✗	✓

Git Basics

Architecture

- Remote: The central repo (on a host machine/server, e.g., Github or Gitlab) → is identified by the alias "origin"
- Repository: The local repo (.git sub-directory inside your working directory), created by "git init" or "git clone", i.e., ceartion/clonining
- Index or staging area: State between the working directory and repository (after modifying and before commiting)
- Workspace or working directory: your local machine, including all directories, sub-directories, and files of your project

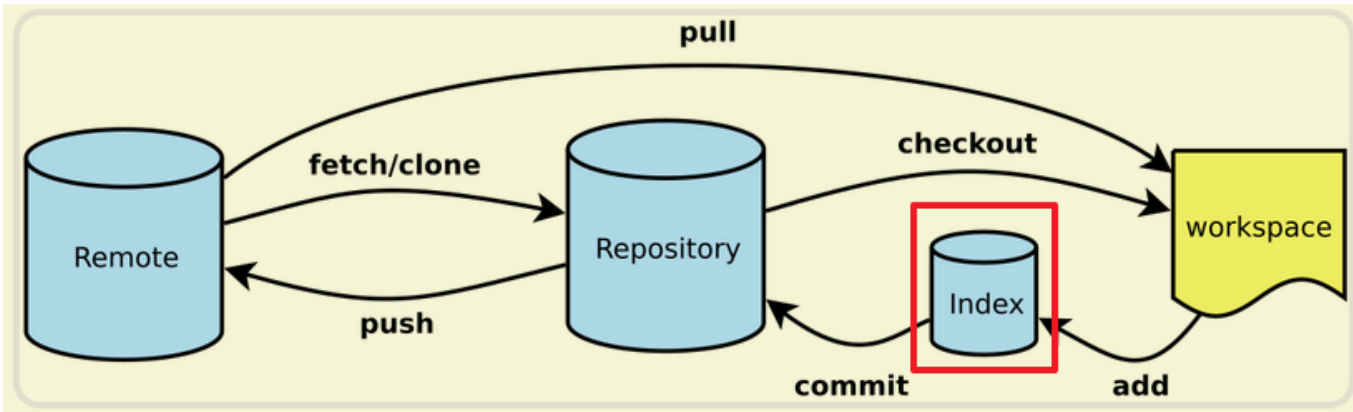

Figure: Git architecture (Source: www.stackoverflow.com)

Git Basics

Definitions

- **origin**: A shorthand name for the remote repo

  $git remote show (shows "origin" as output)

  $git remote show origin (shows detailed info on origin)
- **branch**: A movable pointer to a commit
- **master (or sometimes main)**: Default name of the (first) branch: can be changed
- **HEAD**: A special pointer that tells on (the tip of) which branch you are.
- **origin/HEAD**: A special pointer that tells on which branch the remote repo is.

Git Basics

Add/Commit

- **git add**: To add a new file or modified into the staging (index) area. It makes the changes ready for commiting.

  $git add FILE_NAME

  $git add . (adds all the changes current directory and sub-directories)

- **git commit**: To put the staged files into the (local) repo. Such changes can be tracked, i.e., revert, log, etc.

  $git commit -m "A proper message"

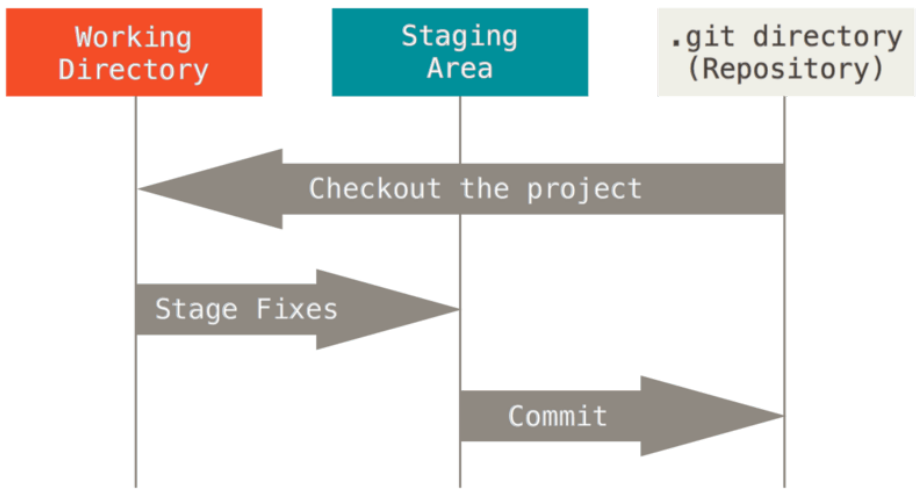

Figure: Git areas (Source: https://git-scm.com)

Git Basics

Initializing a repo

- Creating a local repo (without any remote)

  $git init (creates .git sub-directory)

  $echo "hello world." >> firstFile.txt (makes changes in working area)

  $git status (You see that your commit has some hash value)

  $git add firstFile.txt (puts your changes into staging area)

  $git status (You see that your commit has some hash value)

  $git commit -m "A proper message" (Now you have your first commit on the default branch master)

  Hint: git commit -am "A proper message" (combines "git add" and "git commit")

  $git status (A clean repo and one commit with a hash value)

  $git branch -m master main (renames the branch master to main)

  $git remote (Output is empty since there is no remote repo)

Git Basics

Status and Log

- Status and log

  $git status (Shows the status of the repo)

  $git log (Shows the commit log on the current branch)

  $git log SOME_BRNACH (Shows the commit log on a specific branch)

  $git log --all (Shows the commit log on all branches)

  $git log -p (Shows the commit log and the content difference of files per commit, combines git log and git diff)

  $git log --decorate --oneline --graph --all (Very useful graph-like history)

Git Basics

Aliases

- Git Aliases, some useful examples:

  $git config --global alias.g 'log --decorate --oneline --graph --all' (makes "git g" an alias for the previous long command)

  $git config --global alias.l log (makes "git l" an alias for "git log")

  $git config --global alias.loa 'log --oneline --all' (makes "git loa" an alias for "git log --oneline --all")

  $git config --global alias.s status (makes "git s" an alias for "git status")

  $git config --global alias.b status (makes "git b" an alias for "git branch")

  $git config --global alias.ch checkout (makes "git ch" an alias for "git checkout")

Git Basics

Difference

- Comparing files on the same branch

  $git diff (shows the difference between working and staging area for all files→ to be staged, i.e., git add)

  $git diff SOME_FILE (shows the difference between working and staging area for a given file)

  $git --staged diff (shows the difference between staging area and last commit for all files→ to be commited)

  $git --cached diff (the same as above)

  $git diff HEAD (combines "git diff" and "git diff --staged")

- Comparing files between two branches

  $git branch BRANCH_A..BRANCH_B (compares all files)

  $git branch BRANCH_A..BRANCH_B SOME_FILE (compares only a given files)

Git Basics

Checkout

- Go back to some specific commit

  $git checkout 53c5105 (8 first digits out of a 40-long hexadecimal digit HASH-1)
  Hint: Now you get the message HEAD detachhed at 53c5105

  $git checkout master (to reattach the HEAD to master)

  $git switch master (to reattach the HEAD to master)

  $git switch - (to reattach the HEAD to last branch)

  $git checkout 53c5105 SOME_FILE (checks out only the given files to the given commit)

  $git checkout . (checks out everything to last commit)

  $git checkout HEAD . (the same as above)

  $git checkout --FILE_NAME (restore the file to the last commit)

Hidden Area

Stash

Switching between branches or checking out older commits, while having unclean staging area, are avoided or cause conflicts. Staged (but not commited) changes can hidden into the stash area to clean the working area. Then, you can do whatever you want, while changes in the stash remians, until they are applied or deleted.

- `$git stash` (stashes all staged changes)
- `$git stash save "Something"` (stashes the changes with a label)
- `$git stash list` (shows the stash area, might be more than one entry)
- `$git stash apply stash@{1}` (applies the stash before most recent)
- `$git stash show stash@{1} -p` (shows content of 2nd newest stash)
- `$git stash apply` (applies the most recent stash)
- `$git stash pop` (applies and then deletesi the stash)
- `$git stash clear` (clears the stash area)
- `$git stash drop stash@{2}` (deletes the 3rd newest stash)

Branches in Git

Creating, Displaying, and Switching

- Creating a branch
  - $git branch NEW_BRANCH (creates a new branch)
  - $git checkout -b BRANCH_NAME (creates and switch)
  - $git switch -c BRANCH_NAME (creates and switch, from Git 2.23)
- Displaying branches
  - $git branch (shows only local branches)
  - $git branch -r (shows only remote branches)
  - $git branch -a (shows all branches)
- Switching between branches
  - $git checkout BRANCH_NAME (switches to another branch)
  - $git switch BRANCH_NAME (switches to another branch)

Branches in Git

Comparing, Merging, Renaming, and Deleting

- Comparing two branches

  $git branch BRANCH_A..BRANCH_B (compares all files)

  $git branch BRANCH_A..BRANCH_B SOME_FILE (compares only a given files)

- Merging branches

  $git merge BRANCH_B (merges branch b into branch a, you should be in branch a)

- Renaming a branch

  $git branch -m OLD_NAME NEW_NAME

- Deleting a branch

  $git branch -d BRANCH_FOR_DELETION (deletes a branch)

Remote Branches

Creating and Deleting

- Creating remotes branch in command line, where branch A already exists locally

  $git checkout -b origing/BRANCH_A

- Deleting a remote branch

  $git branch -d –remotes origin/BRANCH_FOR_DELETION

Remote Repository

Clone

- Getting a remote repo:

  $git init

  $git remote add origin REPO_ADDRESS/REPO_NAME.git

  $git pull origin master

  $git branch --set-upstream-to=origin/master

- or the easy way $\rightarrow$ git clone

  $git clone REPO_ADDRESS/REPO_NAME.git

  $git clone /home/ehszandi/Public/gitIntro.git (This is an example)

Now the repository is cloned and you can work on it!

Remote Repository

Remote

- **origin**: A shorthand name for the remote repo

  $git remote show (shows "origin" as output)

  $git remote show origin (shows detailed info on origin)

  $git remote -v (shows push/fetch address of the origin)

Remote Repository

Fetch, Pull, Push

- To fetch the last changes in the origin/main

  $git fetch origin (brings your HEAD one commit behind origin)

  $git merge origin/main (to merge origin/main into your HEAD)

  $git pull origin (fetches and merges at the same time)

  $git push origin/main (pushed the last changes in your HEAD into origin/main)

Undoing Changes

Restore

- Unmodify changes (to the last commit)

  $git restore FILE_NAME

- Restore one file to three commits prior to HEAD

  $git restore source=HEAD~3 FILE_NAME

  $git restore FILE_NAME (unstage the file)

  $git restore --staged FILE_NAME (unstage the file)

Undoing Changes

Reset

- Unsaging a file

  $git reset HEAD SOME_FILE

- Becareful: This command removes the history.

  $git reset <hash-commit>

  $git reset HEAD~3 (moves the HEAD to 3 commits before current, but keep the changes) Careful: The last 3 commit will be removed

  $git reset--hard HEAD~3 (moves the HEAD to 3 commits before current and do nor keep the changes) Be even more careful

Alternative: "git revert", which instead deleting intermediate commits, brings the commit of desire forward and creates a new commit upon merging. "git reset" may be used very carefully, only you want to reset to some prior commit, since which you have not pushed anything into the remote.

Undoing Changes

Revert

- syntax for revert is very similar to reset. They differ only in the way things are done

Git Reflog

reflog

- Reference logs, or "reflogs", record when the tips of branches and other references were updated in the **local** repository.
- It expires by default after 90 days (default can be changed)
- HEAD@{0} means the very first reflog, while HEAD@{1} means the second and so forth.
- What comes inside @{ } is called **qualifier**, i.e., entry number or time

$git reflog show (shows .git/logs/HEAD in reverse order)

$git reflog show HEAD@{0} (the same as above as, up to the latest reflog, i.e., HEAD@{0})

$git reflog show HEAD@{2} (does not show the first two reflogs)

$git reflog show HEAD@{27.Aug.2021} (reflogs up to the given date)

$git reflog show HEAD@{one.week.ago} (reflogs up to last week)

$git reflog show master@{one.week.ago} (reflogs up to last week, but only for master branch or any other branch)

Git Reflog

reflog

- Hint: While HEAD~2 means 2 commit ago, HEAD{2} does not mean that. If you switch to another branch, it is inserted in reflog.

  $git checkout HEAD~2 (checks out the 2 commits before the tip of the current branch)

  $git checkout HEAD@{2} (it could a commit on another branch, or even it could the same commit as HEAD, but only refereing to switching between branches)

Git Reflog

reflog

- Saving the mistakes or undoing the past: Find the hash number of the state to which you want to go back using "git reflog show", then do one the followings:

  $git reset --hard 34a523fc

  $git reset --hard master@3

  $git reset --hard HEAD@4

- **Limitation:** git reflogs are **only** local

Rebase

Rebase

- Rebase has two usage:
  1. Using rebase as alternative to merge, which also cleans simplifies/cleans the log graph. The workflow is the same as "git merge", but instead with "git rebase" command.

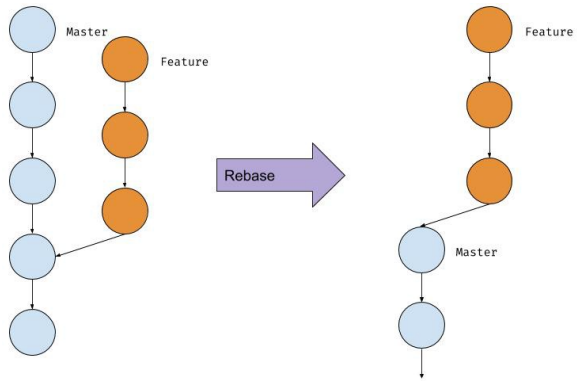

Figure: Regular architecture (Source: https://itnext.io)

  2. Interactive rebase to modify/delete commits

     $git rebase -i HEAD~3 (It starts editing the last 3 commits)

Tag

Create

- Tags are to refer to (important) moments of time
- They are used, e.g., as reference to release versions
- There two types of tags
  1. **Lightweight** tag which is just a pointer to a specific commit.

     $git tag -a TAG (lightweight tag the current commit)

     $git tag -a TAG HASH (lightweight tag the given hash)
  2. **Annotated** tag which is checksummed, contains the tagger name, email, and date; have a tagging message; and can be signed and verified with GNU Privacy Guard (GPG).

     $git tag -a TAG (annotated tag the current commit)

     $git tag -a TAG HASH (annotated tag the given hash)

Tag

List, Move, Delete, Push

- Tags can be used as argument to other git commands, such as *diff*, *checkout*

  $git tag (shows all existing tags)

  $git tag -l "v17*" (shows all tags starting with v17)

  $git tag -f TAG (forces move existing tag to the current commit)

  $git tag -d TAG (deletes the tag)

- Git does not push the tag by default to the remote. You should tell git to do so, using

  $git push --tag

Git Objects

Each Git object has its own hash value and is stored in .git/objects/FOLDER/FILE, where FOLDER is the first two digit of its hash value and the FILE Is the rest of its 38 digits. Git has 4 types of objects

1. **commit**: is created upon each commiting and contains a tree object, parent, author, commiter and a message.
2. **tree**: is similar to directory in UNIX and is indeed staging area (index) and contains blobs and sub-trees.
3. **blob** (binary large object): is the file content without file name.
4. **tag** (annotated)

Git Objects

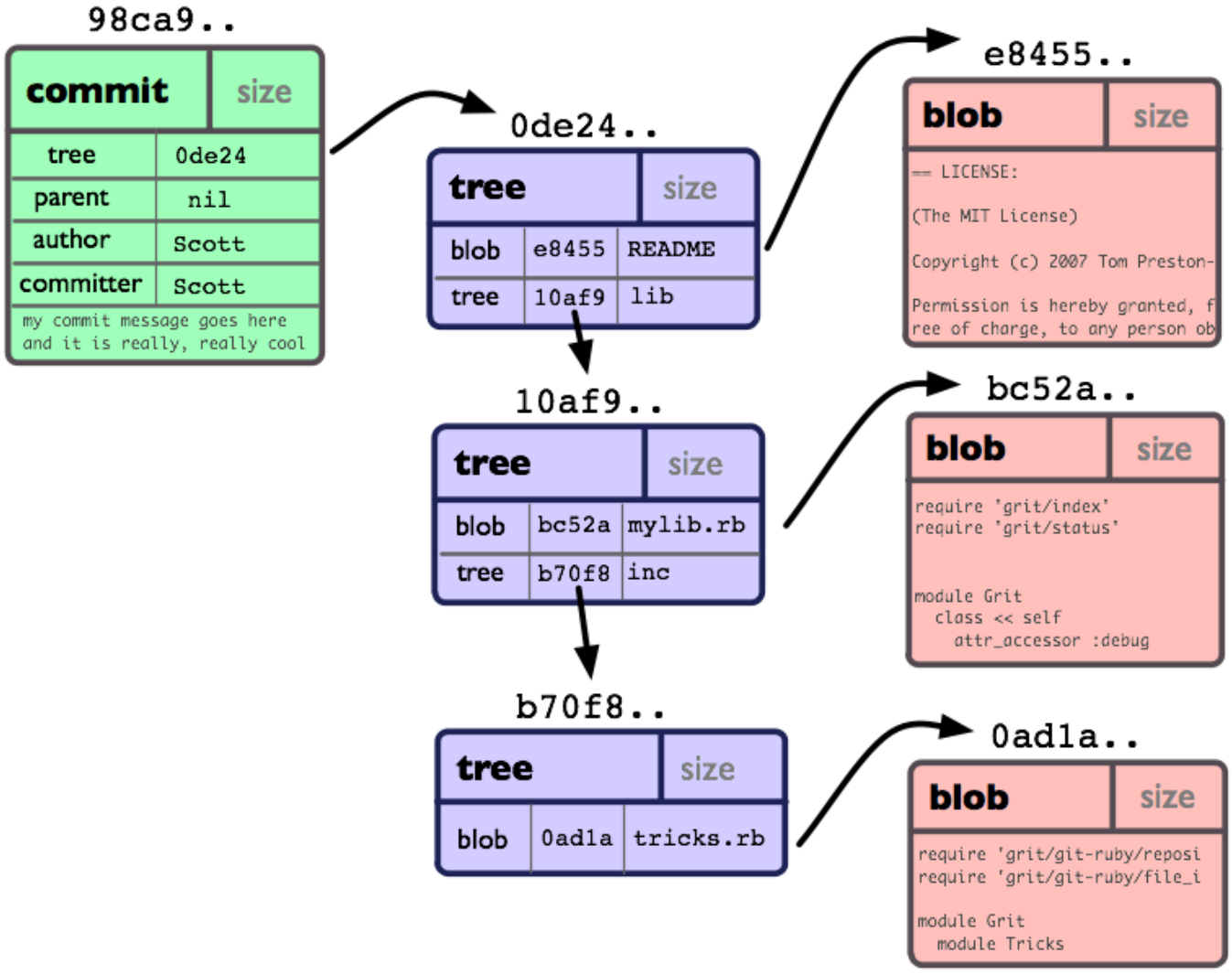

Figure: Commit, tree and blob objects (Source: https://shafiul.github.io)

Git Objects

- Create hash object

  $git hash-object -w FILE (-w stands for write. If given, saves the hash .git/objects. If not given only shows the hash)

  $echo "hello" | git hash-object --stdin -w (created hash from standard input)

  $git cat-file -t HASH (shows the type of hash value)

  $git cat-file -p HASH (shows the content of hash value)

  $git cat-file -p BRANCH^{tree} (shows the content of tree, i.e., blobs and sub-trees, of the tip of the given branch)

  $git cat-file -p HASH^{tree} (shows the content of tree, i.e., blobs and sub-trees, of the tip of the given hash)

Git Objects

Add, Commit in low level Git

This examples shows how to create, add and commit changes into git, without using "git add" and "git commit" commands.

- We add one line, i.e., "hello", to the FILE (any given file) and commit it.

  $echo "hello" » FILE (adds "hello" to the end FILE)

  $git hash-object FILE -w (outputs a hash which is used below, i.e., HASH)

  $git update-index --add --cacheinfo 100644 HASH FILE (adds the changes to FILE to a new staging area)

  $git write-tree (writes the staging area out to a tree object and creates a tree object with a TREE-HASH)

  $git commit-tree -p PARENT-HASH -m "MESSAGE" TREE-HASH (does what git commit -m "MESSAGE" does)

Git SVN

Cloning

- **One time import from svn**:

  $git svn clone -T trunk -b branches -t tags -s -A authors.txt -r14542:HEAD https://10.32.93.102:9880/svn/pegaplan

- See all local and remote (svn) branches

  $git branch -a

- Checkout a remote (svn) branch, e.g., 9.0, into a corresponding local branch, e.g., 9.0

  $git checkout -b 9.0 remotes/origin/9.0

- Check if the local branch is tracking the remote branch

  $git svn dcommit -n (-n does not actually commit, it is for dry-run)

- After updating the local repository, push it into the remote branch

  $git svn dcommit –username="ehszandi"

Git SVN

Cooperation in Branches

- If the branch is already created, only check it out (see above), otherwise
- Create a remote branch for debug/cooperation with colleagues and switch to the local corresponding branch

  $git ch -b NEW_BRANCH remotes/origin/NEW_BRANCH

- After updating the local repository, push it into the remote branch

  $git svn dcommit –username="ehszandi"

To be learnt

$git update-ref

$git rev-parse BRANCH|TAG|HEAD (gives the proper hash of it)

$it config rerere.enabled true ("reuse recorded resolution" tells Git to remember how you've resolved a hunk conflict so that the next time it sees the same conflict, Git can resolve it for you automatically.)