Projektowanie złożonych systemów telekomunikacyjnych

Basic GIT usage

Aleksander Miera

02 2023

Agenda

1. Overview of version control systems
2. GIT overview
3. Conventions and basic commands
4. Working on branches
5. Bag of useful tricks
6. Exercises

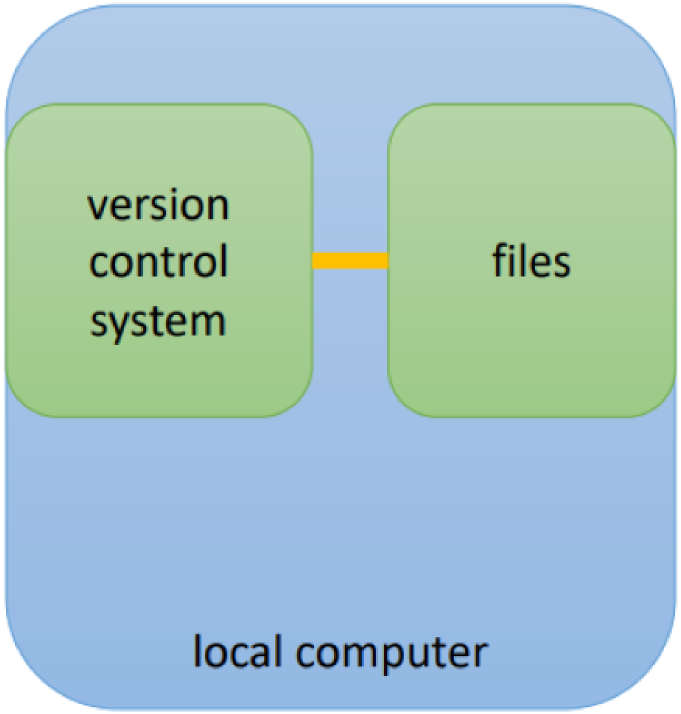
External materials and recommended reading

- GIT online docs:
  https://git-scm.com/docs
- GIT's built-in help, `git help command_name`
- GIT cheat sheet:
  http://rogerdudler.github.io/git-guide/files/git_cheat_sheet.pdf
- Pro GIT, Chacon S., Straub B.
  https://git-scm.com/book/en/v2
- Learn GIT branching:
  https://learngitbranching.js.org/
- Visualizing GIT
  https://git-school.github.io/visualizing-git/

Version Control Systems: basic definitions

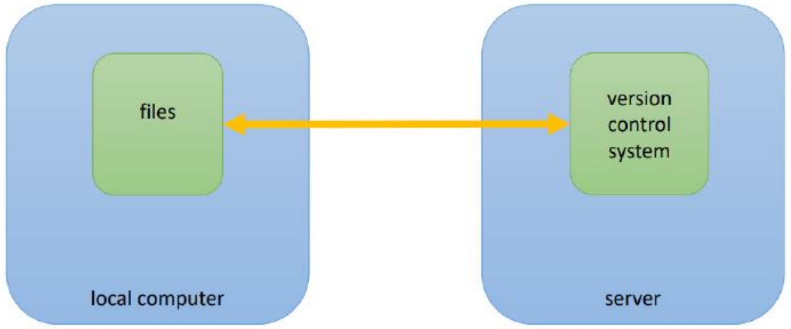
- VCS: Software used to track and manage changes made to the code, in order to enable easier way of developing software.
- Types of VCS:
  - Local
  - Centralized
  - Distributed

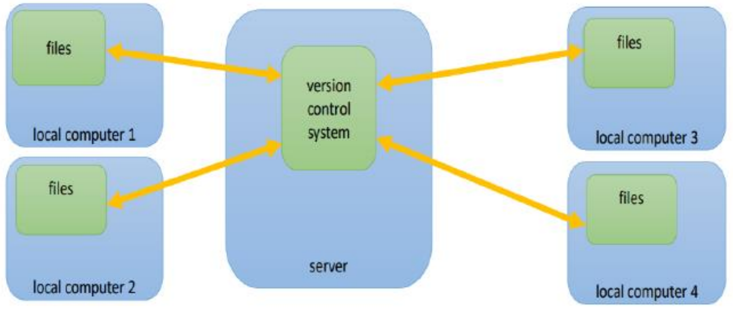
Local VCS



- ☐ Risk of data loss
- ☐ Difficult collaboration between different people

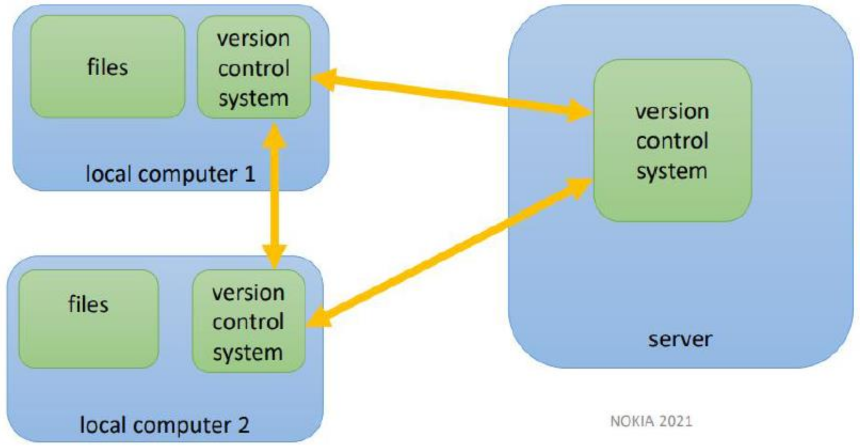
Centralized VCS





- Easy collaboration
- Troublesome when server malfunctions or network access is lost

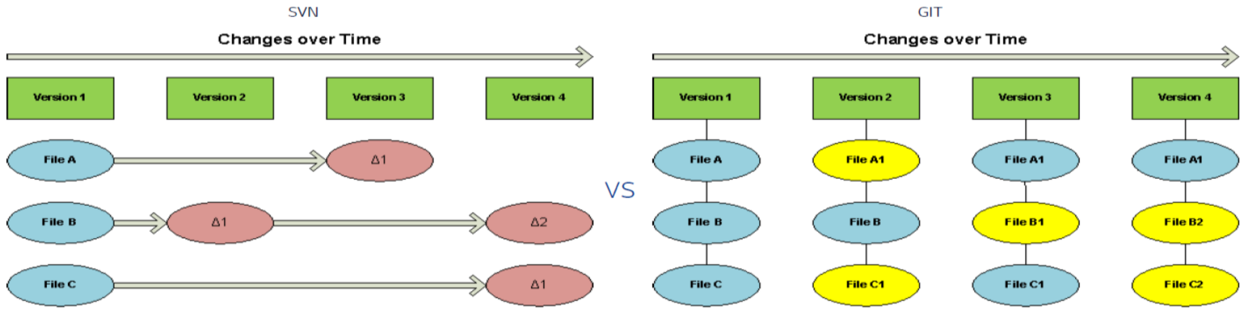
Distributed VCS



- Local copies of the whole repository
- Synchronization between remote server possible on demand
- Most flexibility while working in team

GIT overview

- ☐ Fast
- ☐ Easy to use
- ☐ Collaboration supported out-of-box
- ☐ Distributed
- ☐ Commits stored as snapshots, not diffs



GIT: cookbook for the busy

1. Get the repo: `git clone remote_repo local_path`
2. Create new branch: `git checkout -b new_branch`
3. (write your code here)
4. Check what’s been modified: `git status; git diff`
5. Add modified files to staging area: `git add file_one file_two`
6. Confirm changes: `git commit`
7. Repeat 3-6 as many times as needed
8. Merge/rebase changes:
   1. Merge: `git merge master`
   2. Rebase (rebase to branch=move to the top of branch): `git rebase master`

GIT: cookbook for the busy

9. Send to server: `git push`

10. Sync remote: `git pull`

GIT: basic conventions

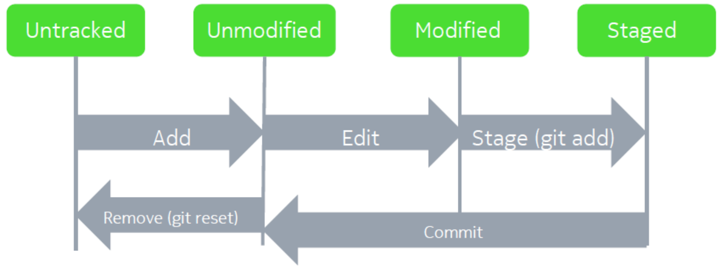
- **HEAD** is the currently checked-out commit
- Branch name is generally an alias to commit's SHA and can oftentimes be used interchangeably.
- ~ is used to indicate the previous commit. ~n is used to indicate n previous commits. Thus: HEAD~ == HEAD~1== "the commit before", HEAD~2 is "two commits before", and featureX~5 is "the fifth commit from the tip of featureX".
- ^ works in a similar manner, but horizontally, i.e. when a commit has multiple parents, i.e. it is a merge commit it ^1, ^2 etc. can be used to distinguish subsequent parent commits. For a single parent ^ == ~
- The general git syntax is:
  ```
  git --general_switch action --action_specific_switch [refspec] --
  [files_to_perform_action_on]
  ```
  It is highly encouraged to try adding files to commands like `checkout`, `reset`, `status`, `diff` , `log` , `show`.

GIT: setting up for work

- ☐ Configure first:
  - ☐ git config --global user.name 'John Doe'
  - ☐ git config --global user.email 'jdoe@gmail.com'
  - ☐ git config --global merge.tool vimdiff
- ☐ Set up SSH keys locally and move the public key to the repo (or setup different kind of authentication if needed).
- ☐ Get the repo from remote server:
  - ☐
    ```
    git clone <repository_url> [optional_local_path_to_cloned_repo]
    git clone git@github.com:alagner/Nokia-PK-2023.git
    ```

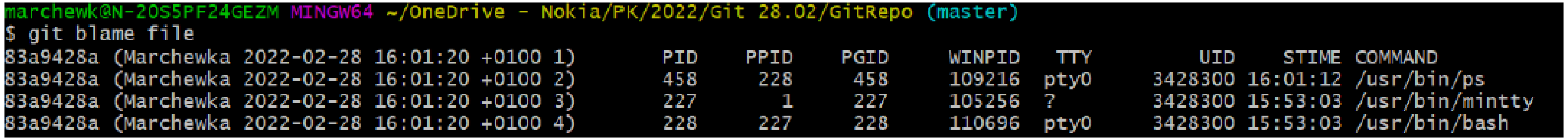
GIT: tracking modifications

- ☐ Inspect repo status according to the following chart:
  `git status`
- ☐ Show diff:
  `git diff` (note: this shows only modified files).
  To see diff of the staging area, use:
  `git diff --cached`
- ☐ Diff between branches:
  git diff branch1..branch2 (homework: .. and ... are different. Read and experiment on that)
- ☐ Open diff in editor:
  git difftool [optional params same as for diff]
- ☐ Show commit history:
  `git log`
- ☐ Show specific commit:
  `git show commit_sha_or_branch`



GIT: who did this?

- Show file split into lines, each with the last person that modified it:
  `git blame`
- Ignore whitespace:
  `git blame -w`
- Limiting lines between 1 and 30 in a given file:
  `git blame -L1,30 filename.cc`

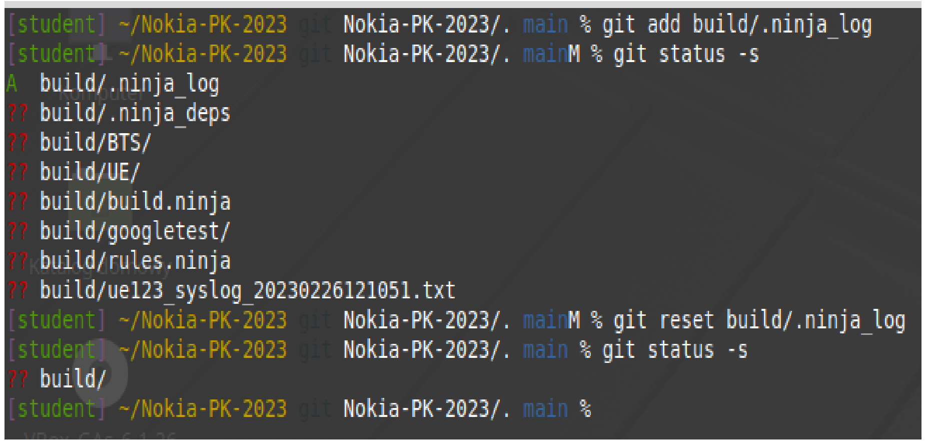


```
marchewk@N-20S5PF24GEZM MINGW64 ~/OneDrive - Nokia/PK/2022/Git 28.02/GitRepo (master)
$ git blame file
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 1)       PID    PPID    PGID     WINPID   TTY         UID    STIME COMMAND
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 2)       458     228     458     109216  pty0     3428300 16:01:12 /usr/bin/ps
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 3)       227       1     227     105256  ?        3428300 15:53:03 /usr/bin/mintty
83a9428a (Marchewka 2022-02-28 16:01:20 +0100 4)       228     227     228     110696  pty0     3428300 15:53:03 /usr/bin/bash
```

GIT: staging area

- ☐ Add modified file to staging area:
  `git add file.txt`
- ☐ Or a whole directory recursively:
  `git add .`
- ☐ Remove file from repo (and stage removal):
  `git rm filename.cc`
- ☐ Unstage file:
  `git reset filename.cc`

When in doubt, always use `git status` to check what's happening.



```
[student] ~/Nokia-PK-2023 git Nokia-PK-2023/. main % git add build/.ninja_log
[student] ~/Nokia-PK-2023 git Nokia-PK-2023/. mainM % git status -s
A  build/.ninja_log
?? build/.ninja_deps
?? build/BTS/
?? build/UE/
?? build/build.ninja
?? build/googletest/
?? build/rules.ninja
?? build/ue123_syslog_20230226121051.txt
[student] ~/Nokia-PK-2023 git Nokia-PK-2023/. mainM % git reset build/.ninja_log
[student] ~/Nokia-PK-2023 git Nokia-PK-2023/. main % git status -s
?? build/
[student] ~/Nokia-PK-2023 git Nokia-PK-2023/. main %
```

GIT: creating new commit

- Create new commit from what’s in the staging area:
  ```
  git commit
  ```
- Modify existing HEAD with staging aread (SHA will be regenerated):
  ```
  git commit --amend
  ```
- Add all tracked and staged files to commit:
  ```
  git commit -a
  ```

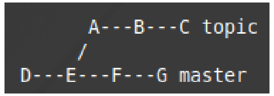
GIT: undoing changes

- Remove commits up to the given one
  `git reset [--soft | --hard | --mixed] commit_sha`
  - Soft: keep the files staged
  - Mixed: keep the files unstage (default)
  - Hard: remove changes (be careful with that!)
- Create revert commit of a given commit:
  `git revert commit_sha`

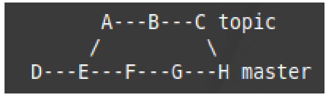
GIT: branches

- List branches
  `git branch`
- Switch branch/go to commit:
  `git checkout commit_sha_or_branch_name`
  - Can be done for specific files, to check-out a specific revision of a specific file do:
    `git checkout commit_sha_or_branch_name -- path/to/file/in/repo/src.cc`
  - The command above can also be used to reset modified, unstaged files to a given state, e.g.:
    `git checkout HEAD -- path/to/file/in/repo/src.cc`
- Create new branch:
  `git checkout -b new_branch_name [optional_starting_point]`
- Delete branch:
  `git branch -d branch_name`
  or to force deletion of a not merged branch
  `git branch -D branch_name`

GIT: branches

- Merge branches
  `git merge branch_to_merge_with`
  Note: this may or may not create a merge commit. Consult docs --no-ff option. TLDR: if a linear way exists from tip of one branch to the other, it only moves the branch pointer.
- For a non-ff case: 

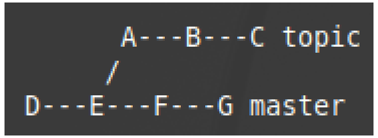
```
      A---B---C topic
     /
D---E---F---G master
```

becomes: 

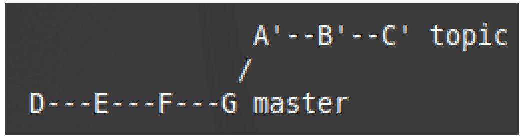
```
      A---B---C topic
     /         \
D---E---F---G---H master
```

GIT: branches

- Rebase. Cut branch and recreate it at a given point/branch
  `git rebase target_point`
  - Assuming that HEAD is at C:
    `git rebase master`
    Will result in:



```
          A---B---C topic
         /
    D---E---F---G master
```



```
                  A'--B'--C' topic
                 /
    D---E---F---G master
```

Note: rebase de facto creates new commits **possibly rewriting history. As a rule of thumb, do not do it on remote branches**. BTW, a lot of other actions can be explained in terms of rebase, e.g. `cherry-pick`, `commit --amend` etc.

Rebase can also be performed interactively:
This is quite complex and goes beyond the scope of this presentation, I encourage you to fiddle with that as a homework.
`git rebase -i target_point`

GIT: branches

- Cherry-pick. Apply a single commit to a given branch:

```
git cherry-pick commit_sha_or_branch_name
```

GIT: merge conflicts

- ☐ While merging/rebasing/cherry-picking merge conflicts can occur:
  ```
  git mergetool
  ```
  should open a text editor (remember the config part earlier on?) with a two or three-way diff view and allow manual conflict resolution.
- ☐ The three files displayed are LOCAL ("ours" side of the conflict, e.g. branch that should contain results of the merge or the one rebased onto), BASE (first common ancestor, not mandatory), REMOTE ("theirs" side, tip of the branch being rebased)
- ☐ After resolving the conflicts, please use `--continue` command line switch with the appropriate action:
  ```
  git merge --continue
  ```
- ☐ Or use `--abort` if resolving the conflicts went south:
  ```
  git rebase --abort
  ```

GIT: collaboration and servers

- ☐ Display available remotes (servers):
  ```
  git remote -v
  ```
- ☐ Get changes and **merge** them into current branch
  ```
  git pull remote_name remote_branch_from:local_branch_to
  ```
- ☐ Get changes and **rebase local ones on top of them**:
  ```
  git pull --rebase remote_name remote_branch_from:local_branch_to
  ```
- ☐ Get changes and **store them in FETCH_HEAD branch** (for further manual actions):
  ```
  git fetch remote_name remote_branch_from:local_branch_to
  ```

  Default pull behaviour can be configured to use rebase instead of merge either for the whole repo or of individual branches. Consult `git help pull` for further information.
- ☐ Put changes onto the server:
  ```
  git push remote_name local_branch_from:remote_branch_to
  ```

GIT: cold things

(i.e. "cool stuff" translated to Spanish and back using Altavista many years ago)

- ☐ Even when deleting commits by `git reset --hard` or `git rebase`, the commits remain until garbage collection is performed. Should things go south, check `git reflog` as a last resort. There is a good chance the old commits are still listed there.
- ☐ Want to send diff to someone? Sure, use:
  `git format-patch commit_from..commit_to`
- ☐ Just got a diff file? Apply it via:
  `git am file_name.patch`
  Already mentioned rules for merge conflicts apply.
- ☐ **Read `git help config` on aliases**

Questions?

NOKIA