Principles of Software Design
GIT and some other stuff

Robert Lukoťka
lukotka@dcs.fmph.uniba.sk
www.dcs.fmph.uniba.sk/~lukotka

M-255

Software configuration management

- Software configuration management is the task of tracking and controlling changes in the software. It includes tracking changes in source code, documentation, and other artefacts.
- Primarily done using Version control systems (VCS).
- Some other systems can be useful in this context (e.g. Issue tracking systems)

What a larger project needs?

- For all artefacts it is known where they are.
- More versions of the same artefacts.
- Multiple people working on the same artefacts concurrently.
- Storing historical versions of the artefacts.
- . . .

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images. Yes.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images. Yes.
- Requirements.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images. Yes.
- Requirements. Yes.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images. Yes.
- Requirements. Yes.
- Deployment scripts.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images. Yes.
- Requirements. Yes.
- Deployment scripts. Definitely.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images. Yes.
- Requirements. Yes.
- Deployment scripts. Definitely.
- Compiler.

What to track?

We should track exactly what is necessary to build, run, and work on the project.

- Manually written source files. Yes
- Generated source files. No - but we need to save the artefacts needed to generate the source.
- Final binary. No.
- Images. Yes.
- Requirements. Yes.
- Deployment scripts. Definitely.
- Compiler. Well, maybe ...

Why you need to store different versions of your software

- You need to fix errors in older releases.
- Different deployment targets (e.g. OS)
- Each historical “version” has its own state.
  - Useful e.g. if you need to track a newly discovered bug.
- Development “versions” of the software.
- . . .

What is a version?

Commit:

- Creates a new version of the system
- Unit of change in VCS
- Each commit should make sense on its own.
- A single commit should not be easy to divide to more commits.
- After a commit the project should remain in a sound state (what sound means varies, e.g. development branch vs mainline branch).

Automated formating tools help make commit diffs more readable. I strongly recommend enforcing them in the project (hooks, continious integration tools).

What is a version?

Branch:

- Separate copies of the system.
- When you commit, you modify only the current branch.
- There are several operations to combine branches.

There are various reasons to have slightly different copies of the system:

- Development branches
- Major releases
- Experimental versions.
- ...

What is GIT?

- Distributed version control
- Created for the development of Linux kernel
  L. Torvalds: I'm an egotistical bastard, and I name all my projects after myself. First 'Linux', now 'git'.
- GITHub - web based version control repository and Internet hosting service – **do not confuse it with git**. Alternatives include GitLab, BitBucket, SourceForge, . . . ,
- GIT is just one particular VCS, there are alternatives too, e.g. CVS, SVN, . . . Some of the above services support other VCS than git.
- Version control services have many other features to manage projects unrelated to git.

Distributed version control [1]

- Clients instead of just taking the versions they need to work, have local repository that can contain everything central VCSserver has.
- There may be more equivalent repositories (there may not be a central server, a decentralized structure may exist).

GIT configuration [2]

There are three main levels of configuration:

- computer level (--system)
- user level (--global)
- project level

You need to set

- Name
- E-mail address

You want to set

- Your favorite text editor to write commit messages and other stuff

GIT configuration [2]

- `git config --global user.name "Robert Lukoťka"`
- `git config --global user.email lukotka@dcs...`
- `git config --global core.editor vim`

Creating a local repository [3]

- `git init`
- `git clone`

File states [4]

- Untracked
- Unmodified
- Modified
- Staged

Basic workflow in local repository [4]

Basic commands:

- `git status`: shows the state of the files
- `git add`: changes the state to staged
- `git rm/mv`: if you delete/move files, let git know
- `git commit` (`git commit -a`)

You may want to do other stuff:

- `git diff` (or use gitk)
- `git reset HEAD <file>`: unstage
- `git checkout (--) <file>`: throw uncommited changes
- `git commit --amend`: change last commit

Viewing commit history [5]

- `git log`: Zillions of options [6]
- `git blame`
- `gitk`

Branches [7]

- `git branch`: shows branches
- `git branch <name>`: creates a branch
- `git checkout <branchname>`: change branch
- `git branch -d <name>`: delete branch

Branches - merging [9]

- `git merge` - merges some other branch into current branch, the merged branch still exists.
    - git tries to merge stuff automatically
    - if he does not know what to do, it lets you resolve the conflicts
    - the new commit has links to two commits (top commits of both branches)

Branches - rebasing [8]

- `git rebase` - alternative to merging
  - gits try to apply the commits in other branch one by one
  - it tries to resolve conflicts
  - if he does not know what to do, it lets you resolve the conflicts (this may happen multiple times during a rebase)
  - the commit history is linear (good for bisecting)

Remote repository [11]

Basic commands

- `git clone`
- `git pull`: Incorporates changes from a remote repository into the current branch
- `git push`

Other stuff

- `git push origin -delete "branchname"`
- `git push --force`:
  - changes commit history
  - do not do this if more people are working on the branch
  - e.g. before merging to master you create a better history, then you need to force push it.
- `git fetch`: just downloads from remote repository, without merging to current branch
- `git remote`: manage repositories.

Very basic workflow

- `git pull`
- repeat
    - make changes
    - `git add`
    - `git commit`
- `git push`

Stash

How to pull while you have uncommitted changes and you do not want to lose them?

- git stash
- git pull
- git stash pop - may create a conflict that needs to be resolved

Stash works like stack, and has many other uses

Moving to past versions

- Each commit is identified by a part of its hash.
- `HEAD`: What we see in the working directory, normally top of the branch, however we can move wherever we want by `git checkout`.
- `HEAD~i`: points *i* commits back.
- `git revert <commit>`: This does not change the history, just adds a new commit.
- `git reset --hard <commit>`: This changes the history.
- `git rebase -i HEAD~k`: interactive rebase is a good tool to adjust history
- `git tag`: tag important commits (version bumps, etc.)

.gitignore

- Used to determine which files should be untracked by default.
- It is good idea to track this file.

GIT Hooks [12]

A way to run custom scripts when certain important actions occur.

- Can be used to block the action
- Client side / sever side
- On commit / on merge / on push / . . .
- E.g. runs automated tests before merge into master, if they do not succeed, merge fails.

Continious integration tools

Services hosting git repositories often provide services to run various tests/checks/actions when interacting with the repository.

- Github Workflows
- BitBucket Pipelines
- ...

You may want to:

- Enforce formatting to have reasonable diffs.
- Run tests to guarantee soundness (static code analyzers, automated tests)
- Build deliverables
- Deploy deliverables
- ...

Pull request

Pull requests are a common way to manage development

- The contributor pushes a branch (into a repository or its fork).
- Requests that the project maintainer (or whoever has rights to perform the operation) to merge the changes into the master
- The reason for the name: the contributor asks somebody to pull his version to become part of the mainline.
- Mostly handled by web based version control repositories, with many additional features.
    - Various automaticaly enforced rules may be set: merge into master only after approving review, automated tests must be green, etc.

Workflows [13]

There are various possible workflows. Example

- master branch
- development branch - merges to master only on important milestones
- feature branches - merges to development branch only when the feature (or an important part of the feature) is finished

Make

Allows to run various commands

- Compared to shell scripts, it checks prerequisites
- You create a file named "Makefile". Basic syntax:

  ```
  goal:  dependencies (files or other goals)
  <tab> command
  <tab> command
  <tab> ...
  ```
- Further examples

Make

Why to use make (or stand-alone automated build)?

- Everybody has his favorite IDE, but the build should work for everybody.
- Minimize dependencies
- Configure build for distinct deployments
- ...

It is very common to generate makefiles

- e.g. `CMake`

Many languages have own tools to automate build (often mixed with dependency management)

Markdown

A lightweight approach to add formatting to text files.

How to initiate a small project

- Initiate version controlling (e.g. git)
- Set up how the project is compiled and build (e.g. Makefile)
- Deployment script
- Basic documentation template (e.g. Markdown)
- Set coding standards, workflows, how quality will be enforced, how automatic testing integrates the workflow . . . (git, makefile, . . .)
- Set up reasonable project structure to attain these goals.

Resources I

- Distributed version control
- Getting Started - First-Time Git Setup
- Creating a repository
- Working with local repository
- Viewing commit history
- Branches
- Merging
- Git tutorial
- Hooks
- Example workflows
- GIT hooks
- Makefile tutorial
- An Introduction to Makefiles
- Mastering Markdown

References I

- Distributed version control
- Getting Started - First-Time Git Setup
- Creating a repository
- Working with local repository
- Viewing commit history
- Git log
- Branches
- Rebasing
- Merging
- Rebase

References II

- Remotes
- Hooks
- Example workflows
- Pull request - Wikipedia