Péter Mileff PhD

# Integrated Systems and Testing

## Version Control Systems II.

Department of Information Science
University of Miskolc

## How to organize your repository?

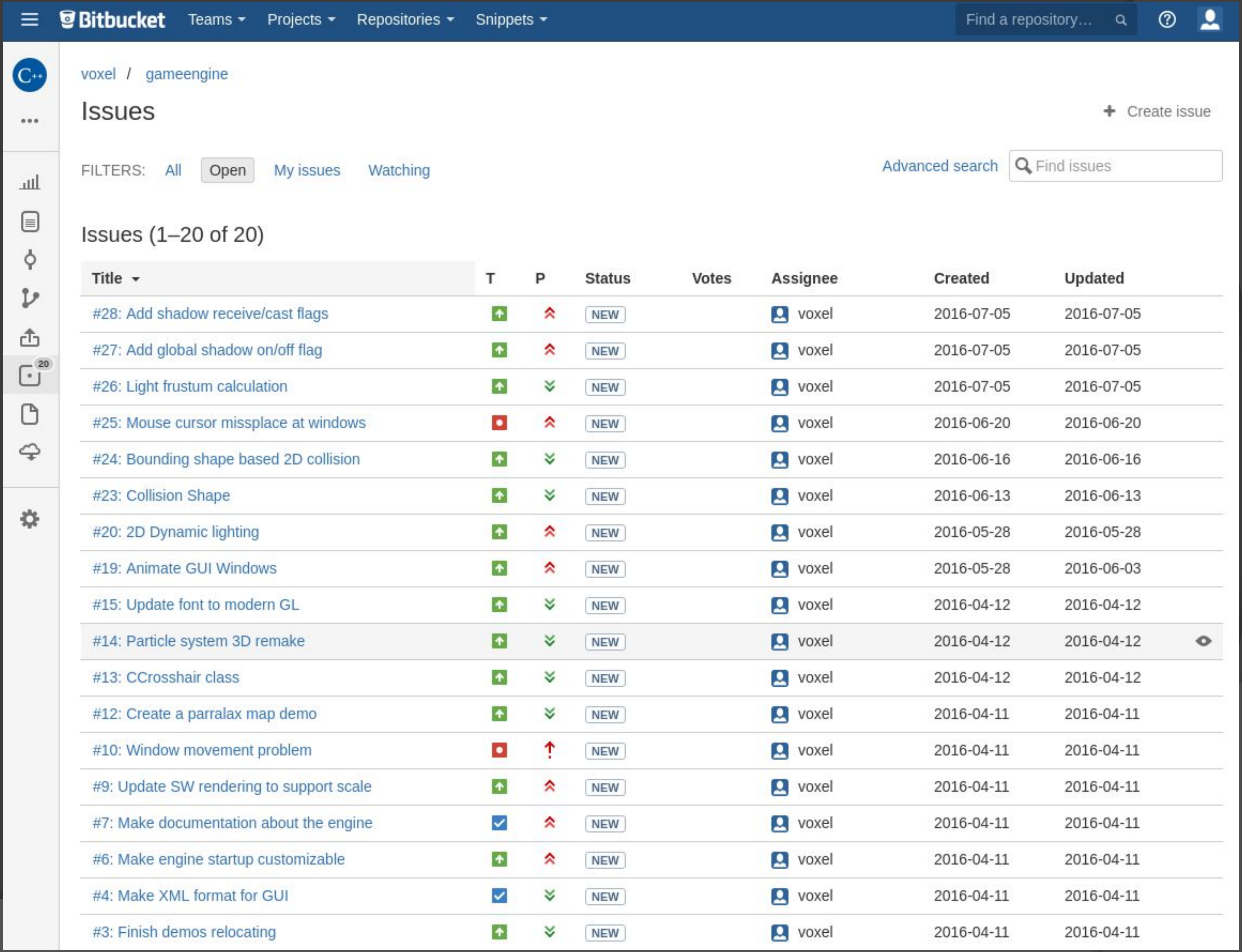
## Key guidelines

- **Today, all major projects are version-controlled**
  - This requires proper repository organization
- **Why?**
  - We need to know who did what and when
  - Code security is important
  - We need to manage:
    - Releases - with name / number
    - Versions - requires proper numbering
    - Development and other branches
    - Other parts / components
  - Link commits with Issue Tracking systems

## An average developer group

- **A typical development process**
  - The development team uses an Issue Tracking system
  - The team organizes problems at least once a week
      - bugs, issues, tasks
    - adds new development tasks/stories
    - The team prioritizes tasks
      - Starts a new development cycle depending on the applied development model
        - e.g.: Sprint
  - Bugs, issues in Issue Tracking are identified by number and name
    - e.g. ISSUE 1357 - Google registration is not working

# Bitbucket example



voxel / gameengine

## Issues

+ Create issue

FILTERS: All [✓ Open] My issues Watching

Advanced search   Find issues

Issues (1–20 of 20)

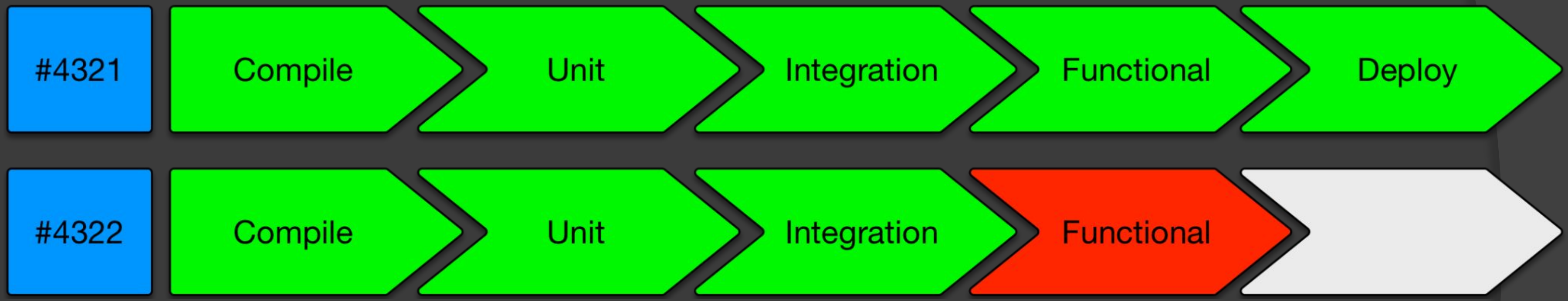
| Title | T | P | Status | Votes | Assignee | Created | Updated |
|---|---|---|---|---|---|---|---|
| #28: Add shadow receive/cast flags | ⬆ | ⇈ | NEW | | voxel | 2016-07-05 | 2016-07-05 |
| #27: Add global shadow on/off flag | ⬆ | ⇈ | NEW | | voxel | 2016-07-05 | 2016-07-05 |
| #26: Light frustum calculation | ⬆ | ⇊ | NEW | | voxel | 2016-07-05 | 2016-07-05 |
| #25: Mouse cursor missplace at windows | 🟥 | ⇈ | NEW | | voxel | 2016-06-20 | 2016-06-20 |
| #24: Bounding shape based 2D collision | ⬆ | ⇊ | NEW | | voxel | 2016-06-16 | 2016-06-16 |
| #23: Collision Shape | ⬆ | ⇊ | NEW | | voxel | 2016-06-13 | 2016-06-13 |
| #20: 2D Dynamic lighting | ⬆ | ⇈ | NEW | | voxel | 2016-05-28 | 2016-05-28 |
| #19: Animate GUI Windows | ⬆ | ⇈ | NEW | | voxel | 2016-05-28 | 2016-06-03 |
| #15: Update font to modern GL | ⬆ | ⇊ | NEW | | voxel | 2016-04-12 | 2016-04-12 |
| #14: Particle system 3D remake | ⬆ | ⇊ | NEW | | voxel | 2016-04-12 | 2016-04-12 |
| #13: CCrosshair class | ⬆ | ⇊ | NEW | | voxel | 2016-04-12 | 2016-04-12 |
| #12: Create a parralax map demo | ⬆ | ⇊ | NEW | | voxel | 2016-04-11 | 2016-04-11 |
| #10: Window movement problem | 🟥 | ↑ | NEW | | voxel | 2016-04-11 | 2016-04-11 |
| #9: Update SW rendering to support scale | ⬆ | ⇈ | NEW | | voxel | 2016-04-11 | 2016-04-11 |
| #7: Make documentation about the engine | ☑ | ⇈ | NEW | | voxel | 2016-04-11 | 2016-04-11 |
| #6: Make engine startup customizable | ⬆ | ⇈ | NEW | | voxel | 2016-04-11 | 2016-04-11 |
| #4: Make XML format for GUI | ☑ | ⇊ | NEW | | voxel | 2016-04-11 | 2016-04-11 |
| #3: Finish demos relocating | ⬆ | ⇊ | NEW | | voxel | 2016-04-11 | 2016-04-11 |

## Version Control Strategies and Continuous Delivery...

## Continuous Delivery

- It is a software development practice in which a team or a company strives to keep their software in a deliverable state at all times.
  - release software to production as often as possible
- Continuous delivery is a prerequisite to continuous deployment
- From a business point of view
  - allows a company to quickly adapt to changing market developments
  - respond to user feedback, etc
- It enforces a large number of software development practices
  - how version control is handled,
  - test and release automation

# Continuous Delivery



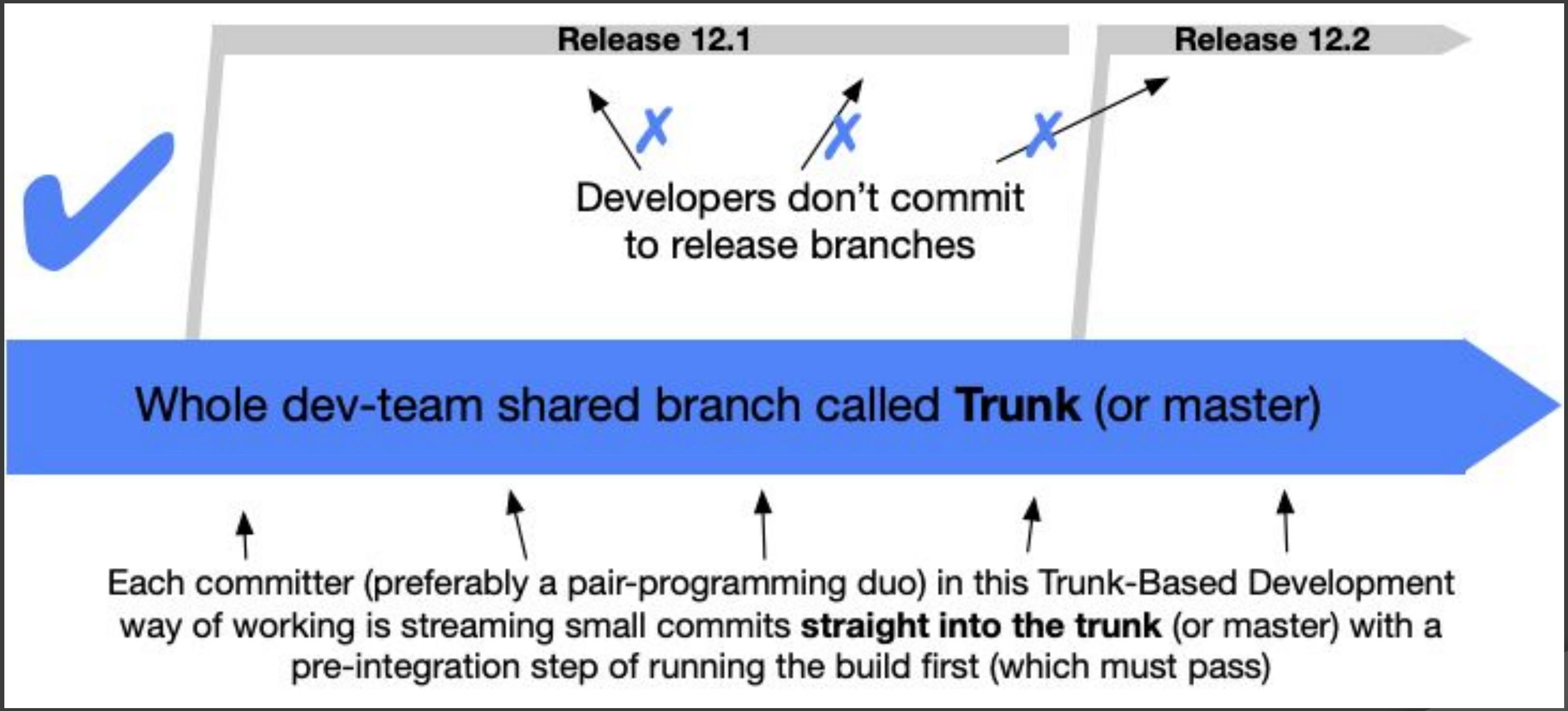
Automatically re-deploy a proven build to a QA or UAT environment

# Trunk-based development

- A source-control branching model
- The simplest way to collaborate with multiple developers on a single codebase
- All developer work on a single branch
  - usually the master branch
- **Heavily implies Continuous Integration:**
  - changes are continuously integrated multiple times per day
  - this keeps everyone up to date on the latest developments
    - new and updated features are quickly known throughout the entire team
  - It intends to prevent people from working on islands, isolated from the rest of the team for more than a day
- Do not create long-lived development branches
  - therefore avoid merge hell

https://trunkbaseddevelopment.com/

## Trunk-based development


Release 12.1
Release 12.2
✗
✗
✗
Developers don't commit to release branches
Whole dev-team shared branch called **Trunk** (or master)

Each committer (preferably a pair-programming duo) in this Trunk-Based Development way of working is streaming small commits **straight into the trunk** (or master) with a pre-integration step of running the build first (which must pass)

https://trunkbaseddevelopment.com/

# Trunk-based development

- **Disadvantages**
  - every push from any developer comes with a risk that they break the build
  - Breaking the build causes the master to be in an unreleasable state
  - It interrupts the continuous deployment flow,
    - and will impede everyone working on the codebase until it has been resolved.

# Trunk-based development

**What are the other branches for?**

- **1. Program Releases (e.g. Play Store):** the software may have multiple releases during its lifecycle.

  E.g. 1.0, 1.2, 2.0, etc
  - Releases must also be managed by the version control system!
  - How?
    - In case of release, we create a branch from the current trunk
    - each release will be a named branch
      - E.g. RELEASE_1, RELEASE_1_1

## Trunk-based development

- Why is a separate branch good for a release?
  - errors discovered in it can also be corrected
    - since the trunk may already contain other functions, so it cannot be used for this
- <u>An error is always fixed in the branch:</u>
    - a) Switch to the current release branch
    - b) Make bugfix
    - c) Commit the fix into the branch
    - d) possibly make a new release
  - If the version in the bug trunk is also affected, it needs to be fixed there as well
    - or merge the change in the release branch back into the trunk

# Trunk-based development

**What are the other branches for?**

- **2. New, big changes:**
  - Some new developments require a separate branch
  - **Why?**
    - Don’t disturb the development of the trunk because it’s a big change
      - most parts of the software will not work
      - inhibits the activities of other developers
    - Often just experimental development
      - Possibly testing new APIs
      - Replacing certain parts, etc.
  - After successful development, the changes are merged back to the trunk

# Feature-based development

**What is a feature branch?**

- A feature branch is a copy of the main codebase where an individual or team of software developers can work on a new feature until it is complete

- Important to have a strategy for how individuals work together:
  - To avoid overriding each other's changes,
    - engineers create their own copy of the codebase, called branches
    - They do not merge their branch until a feature is complete,
      - sometimes working for weeks or months at a time on a separate copy
  - When the feature is ready, the code will be merged into the main branch

## Feature-based development

- Before that merge is made
  - a number of verifications is first performed;
  - the full suite of automated tests is run
  - a code review is performed by peers

**Disadvantage:**

- The long working time can make the process of merging difficult
- because the trunk or master has likely changed due to other engineers merging their branches
- Merging can be problematic
  - there can be a lot of merge conflict

# The seven rules of a great (Git) commit message

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. Use the body to explain what and why vs. how

## The seven rules of a great Git commit message

1. Separate subject from body with a blank line

   not every commit requires both a subject and a body.

   - Sometimes a single line is fine, especially when the change is so simple

E.g.

Sample subject text

Commit body. The explanation of the implemented feature or task.

**The separation of subject from body pays off when browsing the log**

# The seven rules of a great Git commit message

4. Do not end the subject line with a period

- Trailing punctuation is unnecessary in subject lines.

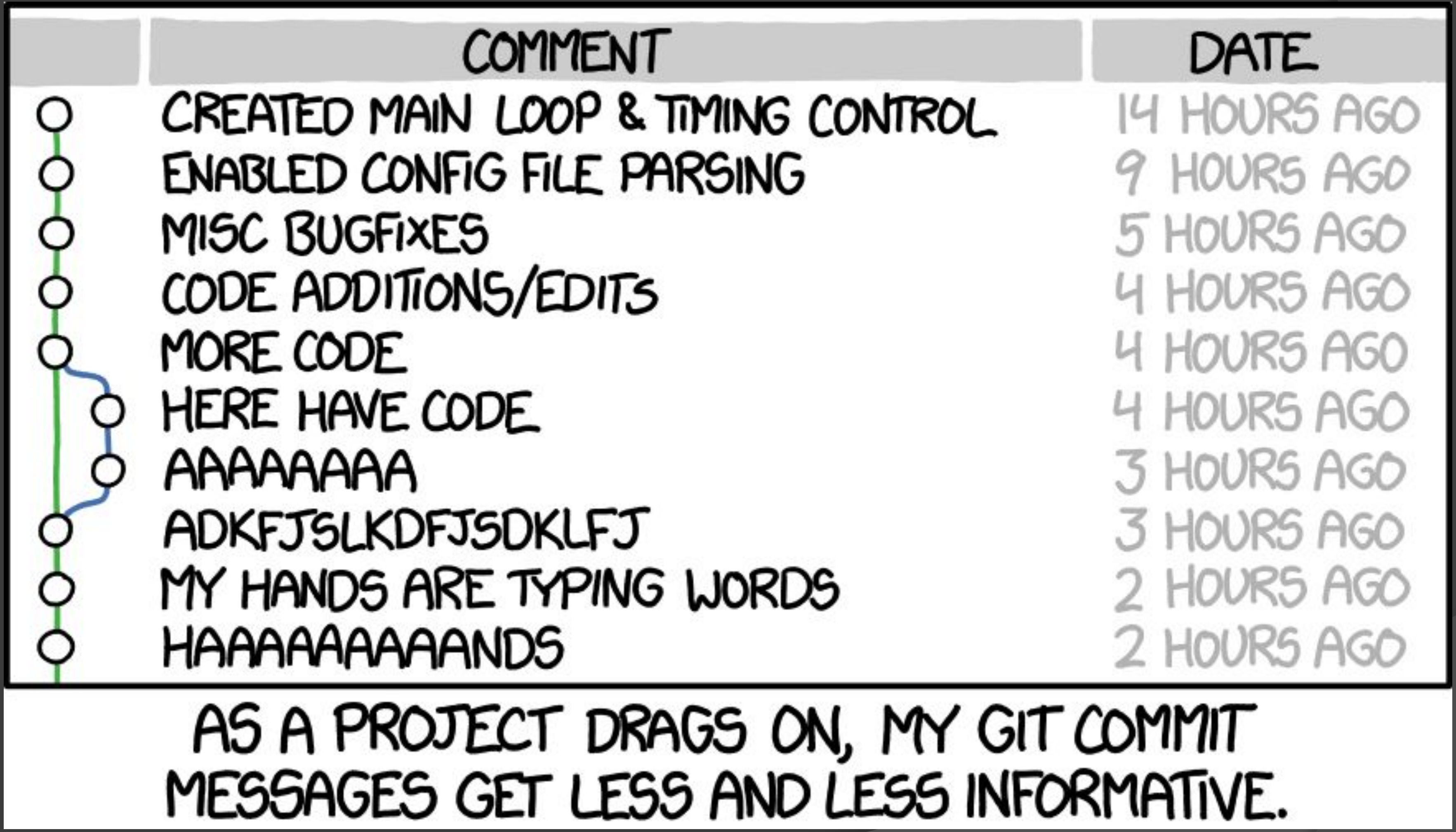
  Example:  Open the pod bay doors
  Instead of:  Open the pod bay doors.

5. Use the imperative mood in the subject line

Imperative mood just means “spoken or written as if giving a command or instruction”. Example:

  Commit: Add google login button

# A bad repository history



| | COMMENT | DATE |
|---|---|---|
| ○ | CREATED MAIN LOOP & TIMING CONTROL | 14 HOURS AGO |
| ○ | ENABLED CONFIG FILE PARSING | 9 HOURS AGO |
| ○ | MISC BUGFIXES | 5 HOURS AGO |
| ○ | CODE ADDITIONS/EDITS | 4 HOURS AGO |
| ○ | MORE CODE | 4 HOURS AGO |
| ○ | HERE HAVE CODE | 4 HOURS AGO |
| ○ | AAAAAAAA | 3 HOURS AGO |
| ○ | ADKFJSLKDFJSDKLFJ | 3 HOURS AGO |
| ○ | MY HANDS ARE TYPING WORDS | 2 HOURS AGO |
| ○ | HAAAAAAAANDS | 2 HOURS AGO |

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

## Software version numbering…

# Version numbering

- The software have many changes during its lifecycle
  - several versions and releases
  - **these should be managed**
- Proper version numbering and interpretation is important!
  - We need because of the development history
    - To be able to go back to a given version
    - To have an exact state
- Primitive versioning and release:
  - the software release is made from the head of the trunk
  - numbering is incremental but there is no deep logic in it

## Version numbering

- There are several versioning schemes
  - There is no “best”,
  - Any of them can be customized to meet the additional needs
- We need to decide the scheme at the beginning of the software development
  - makes development and release logical
  - it does not confuse users either

## Semantic versioning

http://semver.org/

- A widely accepted set of rules
  - Defines the logic of the software versioning
    - detailed, precise
  - Especially for systems:
    - where there are many iterations,
    - releases are frequent,
    - a lot of dependency
  - Typical examples are individual libraries
    - e.g. LibreOffice_5.2.0_Linux_x86-64_rpm.tar.gz

# Semantic versioning

- **Why we need?**
  - To handle package dependency easier
  - To avoid "dependency hell"
- **What is dependency?**
  - It is a third-party bit of software
    - was probably written by someone else and ideally solves a single problem for us
- Dependencies are handled usually by a **package manager**
  - based on semantic versioning
- A software may have several dependency
- **Dependency hell:** several packages have dependencies on the same shared packages or libraries, but they depend on different and incompatible versions of the shared packages

## Semantic versioning

- **A version of a software:**

**Major.minor.patch**

**M.m.p**

Major: Indicates a version change, an API change, that is incompatible with each other

- So, a new code cannot replace the existing without any modification
- E.g. SDL_1.2 < - > SDL_2.0

Minor: means there are new features in API that are backward compatible

- E.g. SDL_1.1 és SDL_1.2

Patch: Patch versions are used for bugfixes. There are no functionality changes. Backward compatible.

- E.g. Facebook Android API: 4.14.0, 4.14.1

## Semantic versioning

- Version numbers are always increased:
  - if M is increased, then m.p will be 0.0 ,
  - if m is increased, then p will be 0, M stays as it is
- The order of the versions is from left to right
  - Examples:

1.2.3 is earlier than 2.1.1, which is also earlier than 2.2.0, which is older than 2.1.6789

## Semantic versioning

- The amount of increment is usually 1
  - If a release goes wrong somehow, we can increase the version number more than 1
    - E.g. release 1.5.3 is built and ready
      - it is broken or have problems
    - A new release can be built, which will be the 1.5.4
    - It should be documented that 1.5.4 will come after 1.5.2

## Semantic versioning

- Semantic versioning also allows for unique names:
  - For example, after the “-” sign, enter pre-release version markings such as alpha1, alpha2, beta9
  - There may be dots in the section after “-”
  - In terms of order, the comparison is still made from left to right in ASCII order
    - So 1.1.0-1 is earlier, than 1.1.0-alpha
  - We can also add build information after the “+” at the end of the version number,
    - but the two versions must not differ only in this

## Mercurial in short...

## Mercurial (HG) usage

- **Create a mercurial repository:**
  1) mkdir project
  2) cd project
  3) **hg init**
- **Add files to repository:**
  1) create hello.txt
  2) **hg add hello.txt**
- Commit:

```
hg commit -m "Add initial version of hello.txt"
```

- Send changes to the server

```
hg push
```

## Mercurial (HG) usage

- **Clone an existing repository**

```
hg clone http://example.com/repo/hello my-hello
```

- **Get the changes from the repo**

```
hg pull
```

- **Apply the changes on the local repository:**

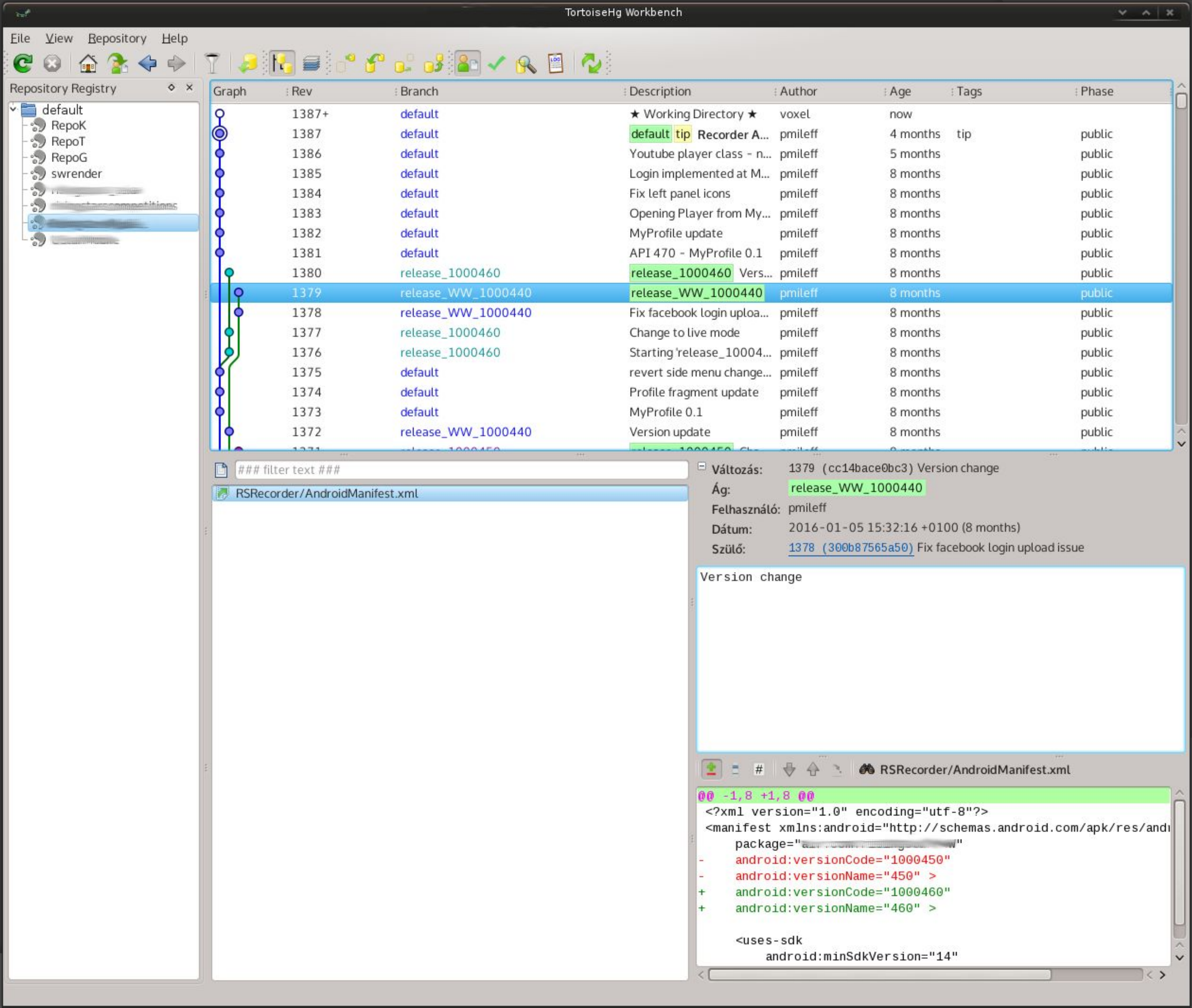
```
hg update
```

- **Merge:**

```
hg merge
```

- **Repo status information:**

```
hg summary
hg log
```



TortoiseHg Workbench

File View Repository Help

Repository Registry

- default
  - RepoK
  - RepoT
  - RepoG
  - swrender
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]

| Graph | Rev | Branch | Description | Author | Age | Tags | Phase |
|---|---|---|---|---|---|---|---|
| | 1387+ | default | ★ Working Directory ★ | voxel | now | | |
| | 1387 | default | default tip Recorder A... | pmileff | 4 months | tip | public |
| | 1386 | default | Youtube player class - n... | pmileff | 5 months | | public |
| | 1385 | default | Login implemented at M... | pmileff | 8 months | | public |
| | 1384 | default | Fix left panel icons | pmileff | 8 months | | public |
| | 1383 | default | Opening Player from My... | pmileff | 8 months | | public |
| | 1382 | default | MyProfile update | pmileff | 8 months | | public |
| | 1381 | default | API 470 - MyProfile 0.1 | pmileff | 8 months | | public |
| | 1380 | release_1000460 | release_1000460 Vers... | pmileff | 8 months | | public |
| | 1379 | release_WW_1000440 | release_WW_1000440 | pmileff | 8 months | | public |
| | 1378 | release_WW_1000440 | Fix facebook login uploa... | pmileff | 8 months | | public |
| | 1377 | release_1000460 | Change to live mode | pmileff | 8 months | | public |
| | 1376 | release_1000460 | Starting 'release_10004... | pmileff | 8 months | | public |
| | 1375 | default | revert side menu change... | pmileff | 8 months | | public |
| | 1374 | default | Profile fragment update | pmileff | 8 months | | public |
| | 1373 | default | MyProfile 0.1 | pmileff | 8 months | | public |
| | 1372 | release_WW_1000440 | Version update | pmileff | 8 months | | public |

### filter text ###

RSRecorder/AndroidManifest.xml

| | |
|---|---|
| **Változás:** | 1379 (cc14bace0bc3) Version change |
| **Ág:** | release_WW_1000440 |
| **Felhasználó:** | pmileff |
| **Dátum:** | 2016-01-05 15:32:16 +0100 (8 months) |
| **Szülő:** | 1378 (300b87565a50) Fix facebook login upload issue |

```
Version change
```

RSRecorder/AndroidManifest.xml

```
@@ -1,8 +1,8 @@
 <?xml version="1.0" encoding="utf-8"?>
 <manifest xmlns:android="http://schemas.android.com/apk/res/andr
     package="[redacted]"
-    android:versionCode="1000450"
-    android:versionName="450" >
+    android:versionCode="1000460"
+    android:versionName="460" >

     <uses-sdk
         android:minSdkVersion="14"
```

GAME OVER