Project Support: Version Control and Issue Tracking

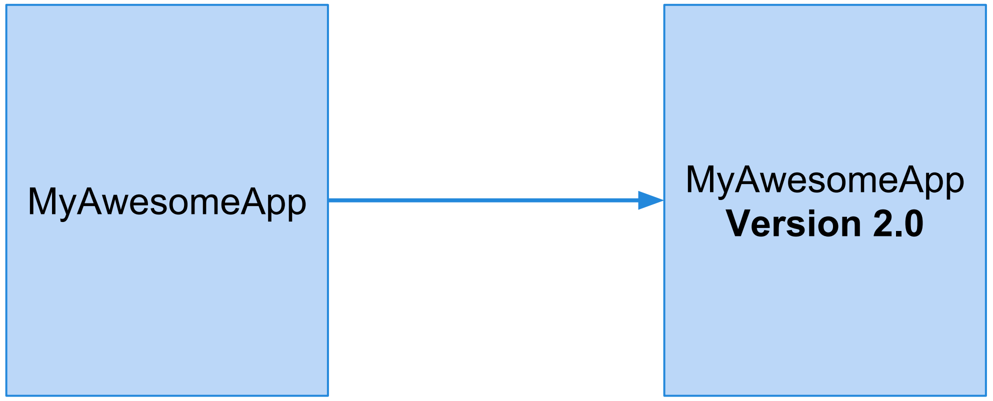
CSCE 247 - Lecture 5 - 02/04/2019

Managing Change

- We now have requirements.
  - Are they set in stone?
  - What needs to happen when they change?
- Next is design, then implementation.
  - What happens when those change?
- These artifacts evolve throughout development, **and** after release.
- Effective engineers can control the impact of change, and ensure that all artifacts evolve when any change.
  - **Today - Version Control and Issue Tracking**

Code Evolution

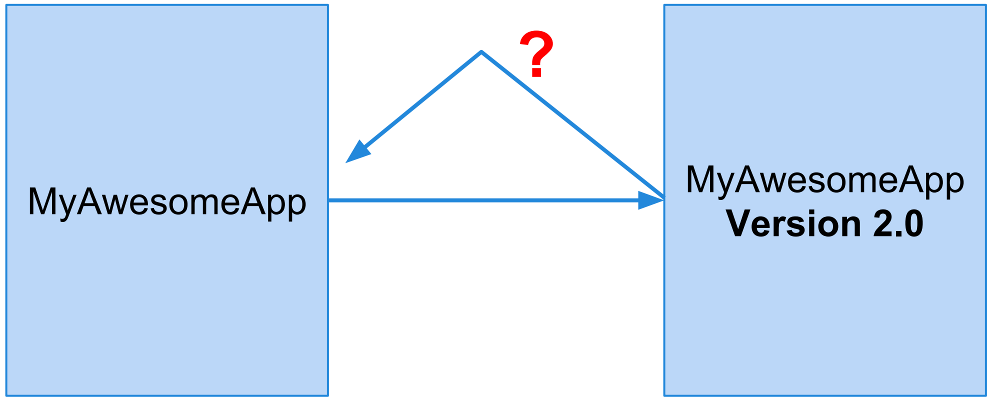
- I want to add a feature to MyAwesomeApp.



- Open in my IDE, edit the code, save.
- **Done, right?**
    - Let’s test it out…

Complicating Factors

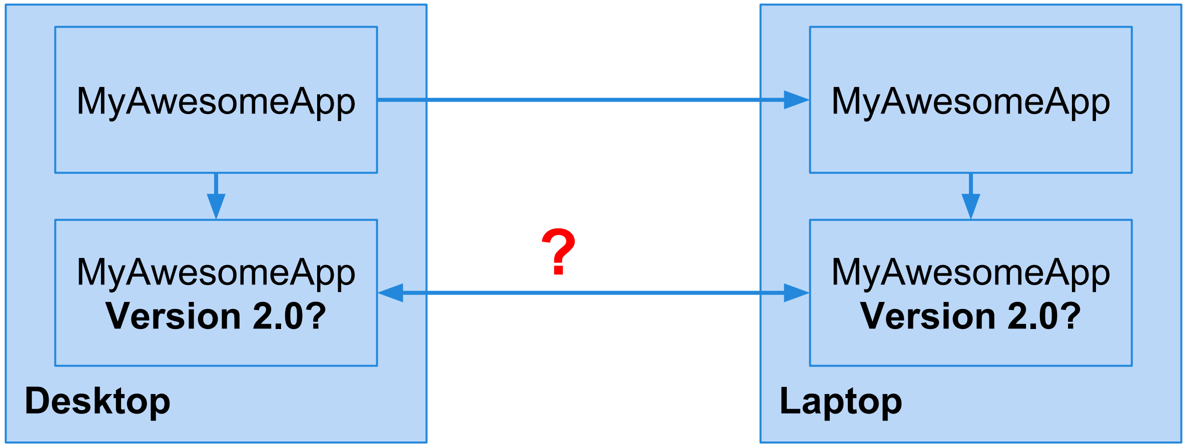
- What if the changes don't work?



- Can we go back to what we had before?
    - Especially hard if we've made a series of changes.
    - When do we overwrite a backup?

Complicating Factors

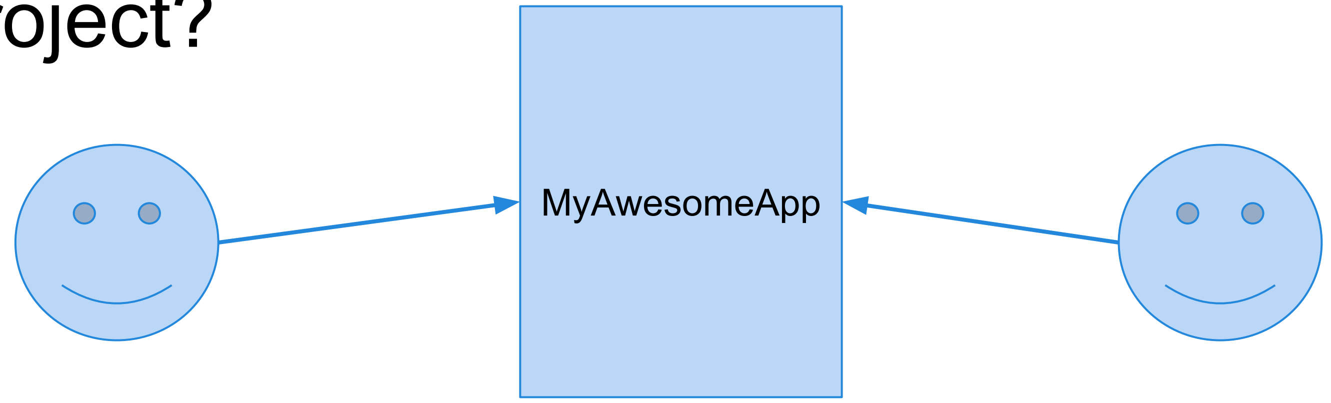
- What if I want to work on my code on two computers?



- Did you copy over all files?
- Did you remember to sync changes?
    - What if you forgot to sync and edited files on both?

Complicating Factors

- What if multiple people are working on the project?



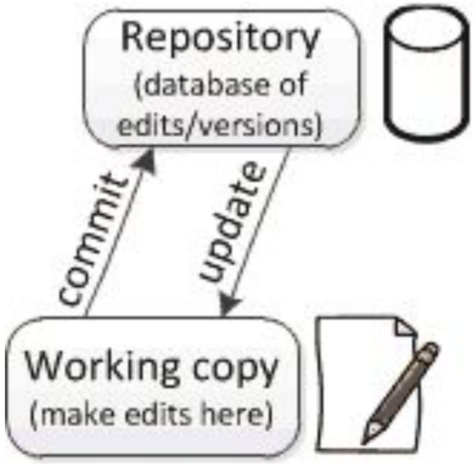
- What happens if multiple people edit the same file?
- When do you synchronize?
- How do you reconcile conflicting changes?

Version Control

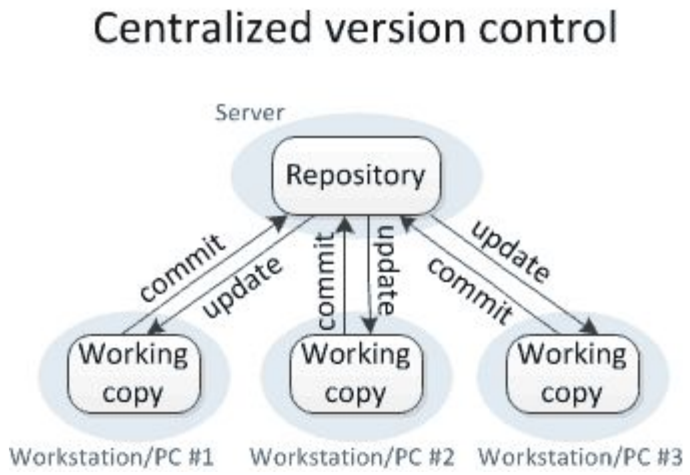
- A version control system (VCS) tracks and manages changes to a filesystem.
  - Tracks addition, deletion, and modification of files.
  - Allows users to share and integrate these filesystem changes to other VCS users.
  - Enables work to progress on multiple machines.
  - Allows backup rollback to previous versions.
- VCS allows collaboration.
  - Each person edits their own copies of files and chooses when to share those changes.
  - Integrates work done simultaneously by team members. Edits to the same file can be combined.

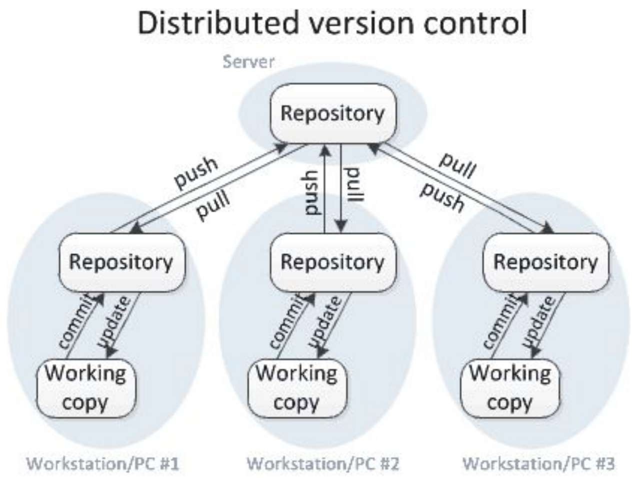
Repositories and Working Copies

- The **repository** is a database of changes made to files.
- The **working copy** is your copy of all of the files in the project.
  - This is where you do your work.
- The repository tracks edits over time, and dictates what the “official” version of the filesystem contains.
  - You update that version by **committing** local changes to the repository.
  - Incorporate changes by **updating** the working copy.



Centralized and Distributed VCS





- In centralized VCS, the repository is stored on a central server.
- In decentralized VCS, all working copies are paired with a copy of the repository.
  - A central repository is used to coordinate and synchronize the local repositories.

Centralized and Distributed VCS

- Decentralized VCS has become standard.
- Centralized CVS has a single point of failure.
  - If the central server is lost, we could lose data.
  - If it is unavailable, developers cannot push changes, merge updates, or roll back code.
- Decentralized avoids this by keeping a copy of the repository in each working copy.
  - If the central server is lost, any copy could become the “central” source of truth.

Basic Workflow

- To establish a working copy, you **checkout** or **clone** the repository.
  - This downloads the current version of all files to your local machine.
    - `git clone https://github.com/User/Project.git`
- Make changes to files.
- You must tell the VCS what you have changed or plan to change.
  - This is called **staging**.
    - `git add MyFile.java`

Committing Changes

- A **commit** takes a snapshot of the project in its current state.
  - The commit command takes the set of changes and writes them to the repository.
  - `git commit -m “Message”`
- Commits form a history of snapshots that can be revisited at any time.
- In decentralized systems, the commit must also be **pushed** to all other copies of the repository.
  - `git push origin master`

Commit Best Practices

- Commits are accompanied by an explanatory message.
  - `git commit -m “Fixes issue #1337 by adding a configuration flag”`
- Use descriptive commit messages.
  - These messages indicate the purpose of the change and are used to find the right commit.
- If the commit relates to a known issue, refer to that issue ID.
- Describe what the commit *does* - use verbs.

Commit Best Practices

- Make each commit a logical unit.
    - Each commit should have a single purpose and should completely implement that purpose.
        - One feature change, bug fix, redesigned class.
    - Makes it easier to locate changes related to a particular feature or bug fix.
    - Allows isolated analysis of changes.
        - Don’t try to fix a fault at the same time you introduce new functionality.
- What if you want to fix an issue before finishing a change?
    - **git commit** can selectively commit files.

Commit Best Practices

- Avoid indiscriminate commits.
  - Specify specific files to commit (either when adding or committing changes).
  - Do not commit all changed files unless you intend to.
  - Do not include temporary debugging changes.
- To avoid committing more than intended:
  - **`git status`** lists all modified files.
  - **`git diff`** shows specific changes made.
  - **`git commit file1 file2 -m “Message”`** only commits the named files.

Commit Best Practices

- VCS is intended for the files people edit.
- Do not commit generated files.
    - .class, .o files. Temporary or compiled artifacts (.pdf).
- Generated files are not needed in version control. Users can regenerate them.
- Generated files are prone to conflict.
- Generated files tend to be binary files. Version control cannot identify differences between two versions of a binary file.
- A **`.gitignore`** file lists filenames to ignore.

Incorporating Changes

- An **update** operation applies the changes made in all commits since your last update.
  - You **commit**, they **update**.
- In decentralized VCS, you must first **pull** updates into your repository, then update your working directory.
  - You **commit**, you **push**, they **pull**, they **update**.
  - Commit and update move changes between working copy and local repository. Push and pull move changes between local and central repositories.
- **`git pull`** performs both pull and update.

Update Best Practices

- Incorporate changes frequently.
    - Run **`git pull`** before making any changes.
        - If someone has made a change before you start to edit, it is a waste of time to make changes then have to resolve conflicts.
    - Run **`git pull`** before **`git push`**.
        - This allows you to resolve conflicts quickly.
- Also push changes frequently!
    - Once you have completed work, share those changes with all copies of the repository.
    - Do not hold changes on your machine.

Conflicts

- A conflict occurs if two users make simultaneous, different changes to the same line of a file.
  - Manual intervention is needed to resolve a conflict.
- Changes 1 & 2 are simultaneous if:
  - User A makes Change 1 before performing the update that brings in Change 2.
  - User B makes Change 2 before performing the update that brings in Change 1.
- Frequent push and pull help avoid conflicts.

Conflicts

- Remember that VCS tools are line-based.
- Be careful with adjusting indentation.
  - Do it if you need to, but check for accidental indentation changes.
- Avoid excessively long lines.
  - 80-character lines are often recommended.
  - Longer lines are more likely to cause conflicts.
    - More people will edit that line.
  - The more characters, the harder it is to determine the exact changes when viewing VCS history.
  - Shorter lines are easier to read when viewing or editing a source file.

Summarizing Basic Workflow

- **`git clone`** (on initialization)
- **`git pull`** (on new change)
- Make local edits.
- Examine local edits:
  - **`git diff`, `git status`**
- **`git add`**
- **`git commit`**
- **`git pull`**
- **`git push`**

Not Just for Source Code

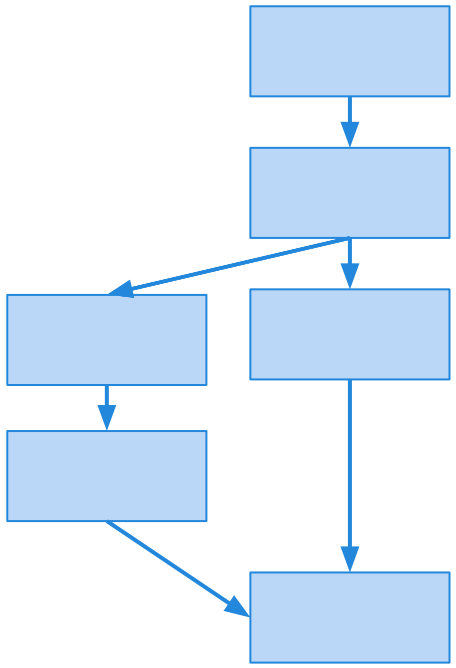
- VCS should always be used for source code, but can also be used for requirements, design, and other documentation.
- Update and commit workflow can be used for any type of file.
    - Binary files (.pdf, .doc, images) are complicated as there is no way to identify specific changes to files.
    - No way to merge elements from versions of conflicting files. VCS simply stores newest version.
    - However, machine-readable files can be used without issue (.txt, .csv).

Branches

- The history of commits forms a **tree**.
- The core version of the system is called the **master branch**.
    - The master branch should be relatively stable, tested, and free of faults (as far as we know).
- How can we add new features or redesign elements of the system while keeping the master branch stable?
    - VCS allows the creation of new branches.

Branches

- Creating a branch takes the current commit of an existing branch (usually master) as the starting point for a new timeline.
    - This can be the current commit to any existing branch.
    - Commits to the new branch are not applied to the original branch.
    - Commits to the original branch are not applied to the new branch.
- When finished, the branch can be merged into the original branch.
    - May require handling conflicts.



Branches

- These are known as **feature branches**.
  - Insulated from master branch, so we can make changes without impacting stability for other developers.
  - Once complete, we can merge the working change into the master branch
- Allows experimentation without breaking anything or impeding others.
  - Many projects have a stable master branch, a development branch, and feature branches stemming from that.
- Enables control over development.

Branches

- To create or switch to a branch:
  - **`git checkout -b <new branch name> [<base branch name>]`**
  - **`[<brase branch name>]`** is optional, default value is **master**.
- To push changes:
  - **`git push origin <new branch name>`**
  - **`Git push origin master`** commits changes to the master branch.
- **`git pull`** incorporates changes to all branches.
- **`git commit`** commits to the currently “checked out” branch.

Forks

- A **fork** is a copy of a complete repository.
  - Contains all past commits and branches.
  - New commits to the original repository are not applied to a fork.
  - New commits to a fork are not applied to the original repository.
    - Commits can be selectively merged into the original.
- Used when a team wants to take the project in an independent direction and does not plan to merge their work.
  - Often used in open-source by amateurs to extend a project they are not part of.

Merging Branches or Forks

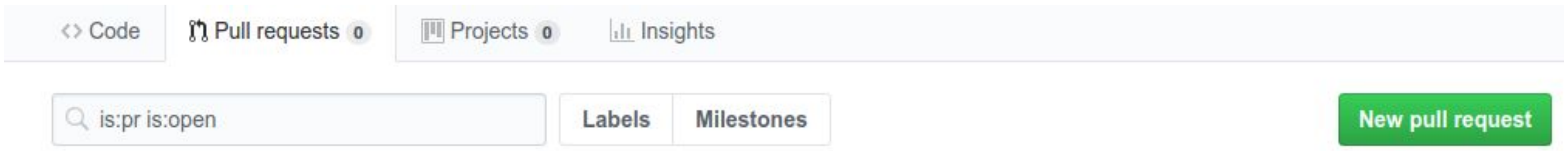
- Pull requests are used to tell others about changes you would like to merge into a branch.
  - Once sent, interested parties can review changes, discuss them, and push additional follow-up commits before agreeing to merge.
  - Performed through the web interface (i.e., Github).
- Once conflicts are resolved, the target branch will be updated by applying all commits to the source branch.

Pull Requests

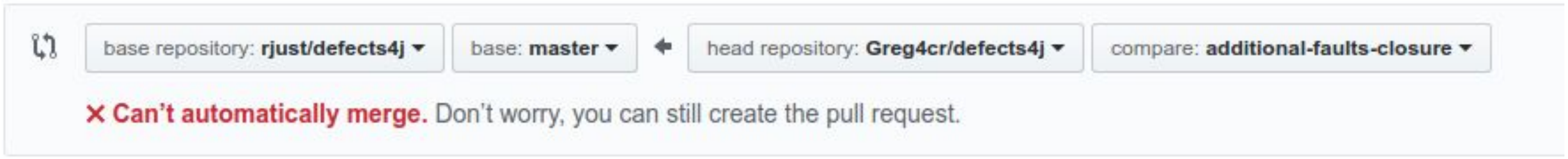
- Common tool in open-source development.
  - Fork the project to create a “personal copy” or create a branch in the core repository.
  - Incorporate your changes.
  - Issue a pull request to the core repository.
    - Same mechanism is used for both fork-to-original and branch-to-branch (i.e., new branch-to-master) merging.
  - Project maintainers discuss the pull request and assess whether it fits the project (and does not break anything).
  - Project maintainers agree to incorporate changes.

Pull Request

- To create a pull request, you must have changes committed to the your new branch.
- Go to the core repository page on Github, click the “Pull requests” tab, and click "New pull request" button.



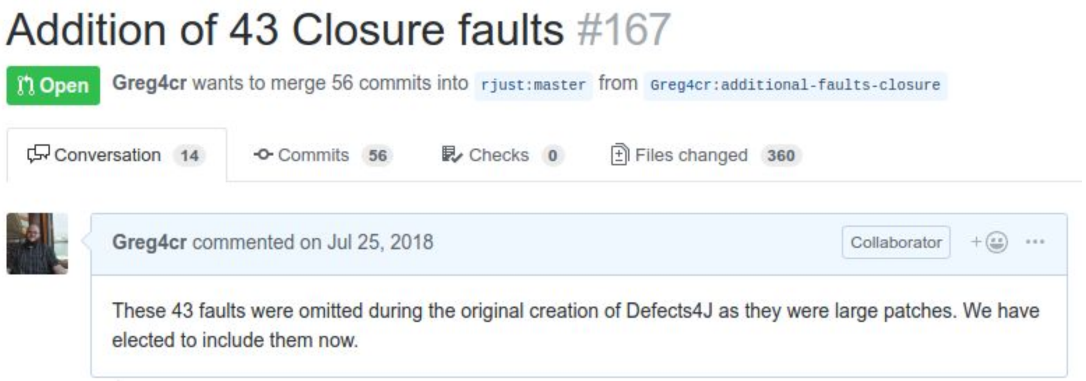
- Pick the source branch using the "head repository” and “compare” dropdowns. Pick the target branch using the “base repository” and “base” dropdowns.


base repository: rjust/defects4j
base: master
head repository: Greg4cr/defects4j
compare: additional-faults-closure

✕ Can’t automatically merge. Don’t worry, you can still create the pull request.

Pull Requests

- Fill out a title and description of the changes.



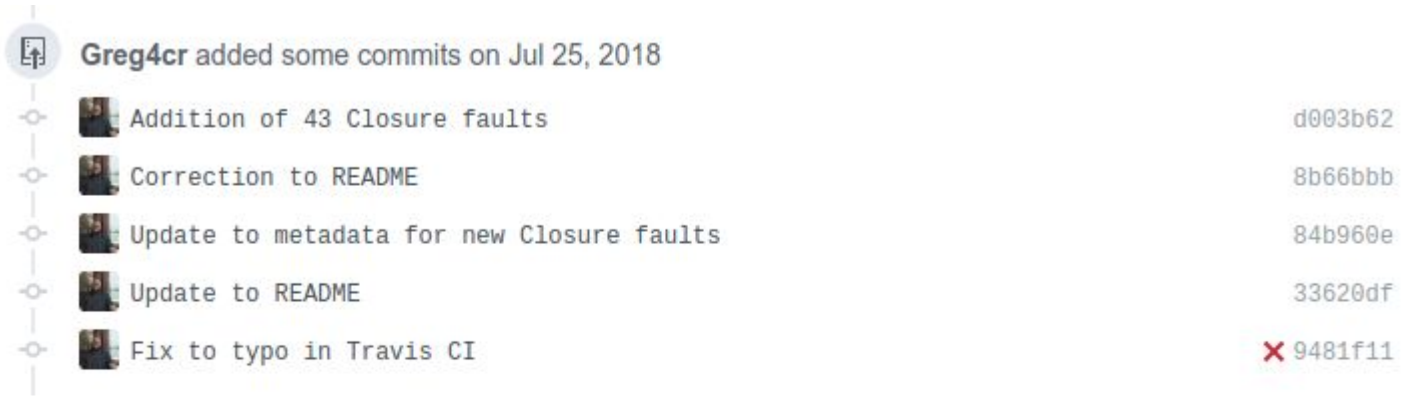
Addition of 43 Closure faults #167

Open Greg4cr wants to merge 56 commits into `rjust:master` from `Greg4cr:additional-faults-closure`

Greg4cr commented on Jul 25, 2018

These 43 faults were omitted during the original creation of Defects4J as they were large patches. We have elected to include them now.

- Commits that will be applied to the target are listed.

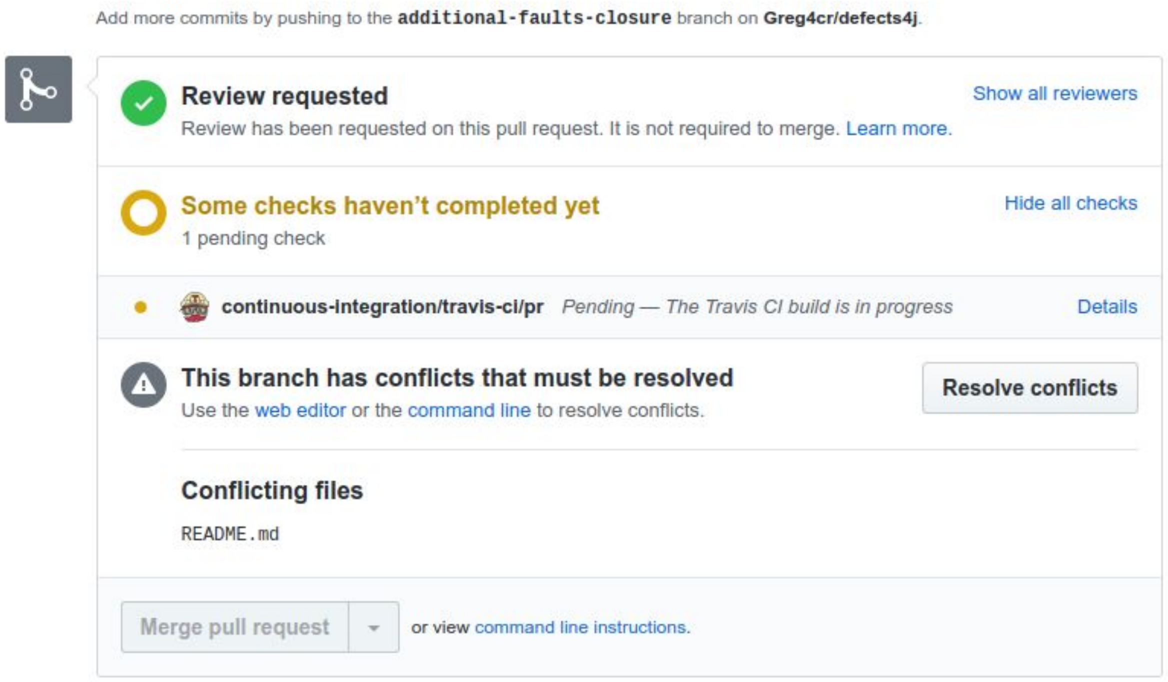


Greg4cr added some commits on Jul 25, 2018

```
Addition of 43 Closure faults                 d003b62
Correction to README                          8b66bbb
Update to metadata for new Closure faults     84b960e
Update to README                              33620df
Fix to typo in Travis CI                    ✗ 9481f11
```

Pull Requests

- Contributors may discuss or review the request.
- Conflicts may need to be resolved.
- When ready, click the “Merge pull request” button.



Add more commits by pushing to the **additional-faults-closure** branch on **Greg4cr/defects4j**.

Review requested
Review has been requested on this pull request. It is not required to merge. Learn more.
Show all reviewers

Some checks haven’t completed yet
1 pending check
Hide all checks

continuous-integration/travis-ci/pr *Pending — The Travis CI build is in progress* Details

This branch has conflicts that must be resolved
Use the web editor or the command line to resolve conflicts.
Resolve conflicts

Conflicting files
README.md

or view command line instructions.

Issue Tracking

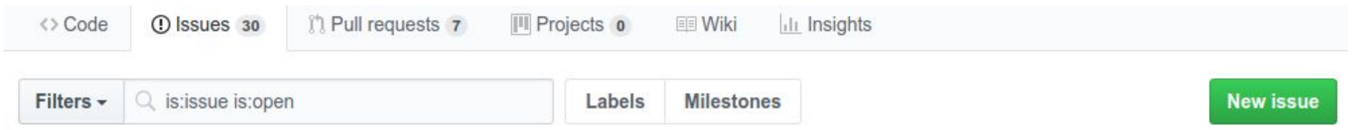
- Issue tracking systems allow developers to track reported software issues.
  - Used to record information about the issue and the status of fixing that issue.
  - Also often used to request new features and track progress on implementation of these features.
- Often available as part of project management software (i.e., Github).
- Issues can be reported by developers and end-users of the system.

Issue Tracking

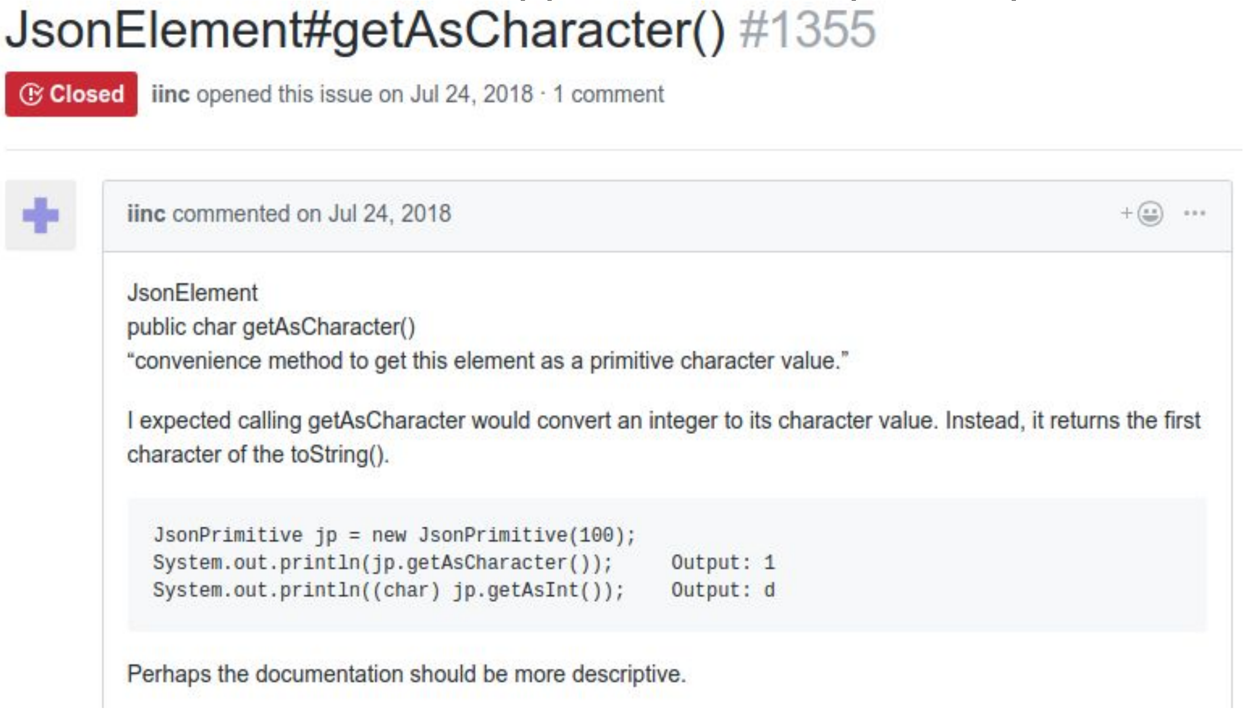
- Commonly tracked details:
  - **What happened:** The erroneous behavior.
  - **How to reproduce the issue:** Steps (and often test cases) that will demonstrate the behavior.
  - **The severity of the issue:** A rating of how badly this affects users (and how widespread it is).
  - **When the issue was reported:** Allows tracking of age of issues, and measuring fix time.
  - **Responsibility:** Who reported the issue and who has been tasked with fixing the issue.

Filing an Issue

- Go to the "Issues" tab on the project page, and click "New issue."



- Fill out a title and description.
  - Be sure to include both what happened and steps to reproduce.


JsonElement#getAsCharacter() #1355

Closed iinc opened this issue on Jul 24, 2018 · 1 comment

iinc commented on Jul 24, 2018

JsonElement
public char getAsCharacter()
"convenience method to get this element as a primitive character value."

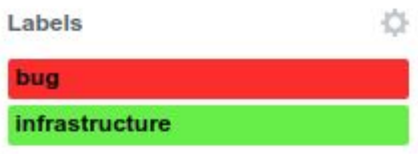
I expected calling getAsCharacter would convert an integer to its character value. Instead, it returns the first character of the toString().

```
JsonPrimitive jp = new JsonPrimitive(100);
System.out.println(jp.getAsCharacter());     Output: 1
System.out.println((char) jp.getAsInt());    Output: d
```

Perhaps the documentation should be more descriptive.

Filing an Issue

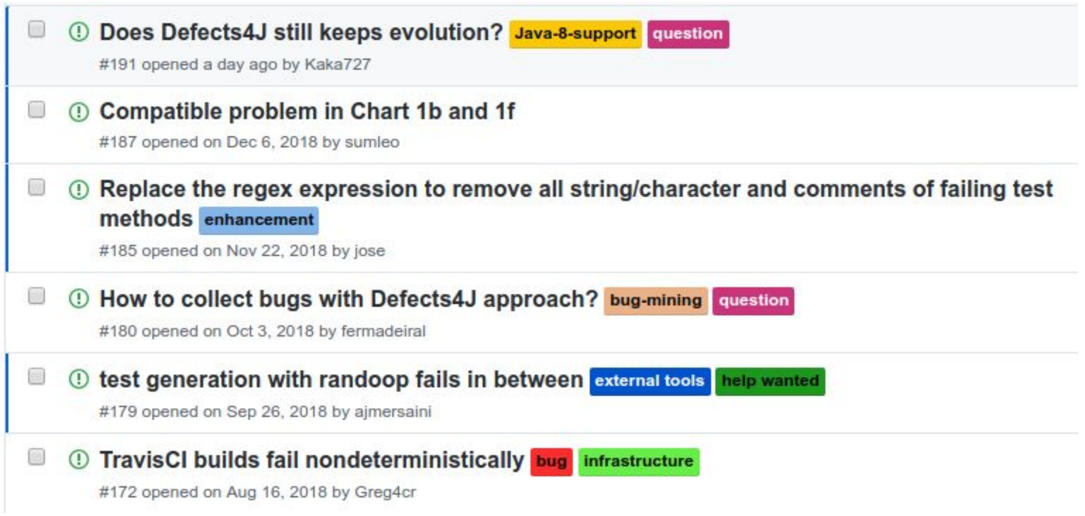
- Issues can be labeled.
  - Common: “bug”, “enhancement”



Labels

bug

infrastructure



Does Defects4J still keeps evolution? Java-8-support question
#191 opened a day ago by Kaka727

Compatible problem in Chart 1b and 1f
#187 opened on Dec 6, 2018 by sumleo

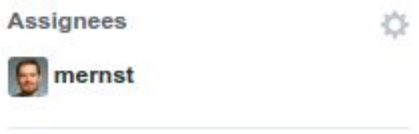
Replace the regex expression to remove all string/character and comments of failing test methods enhancement
#185 opened on Nov 22, 2018 by jose

How to collect bugs with Defects4J approach? bug-mining question
#180 opened on Oct 3, 2018 by fermadeiral

test generation with randoop fails in between external tools help wanted
#179 opened on Sep 26, 2018 by ajmersaini

TravisCI builds fail nondeterministically bug infrastructure
#172 opened on Aug 16, 2018 by Greg4cr

- Users can be assigned to fix the issue.

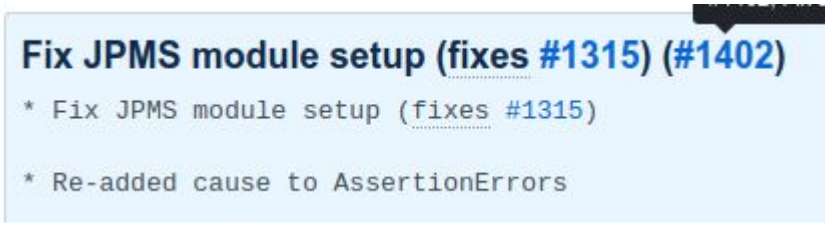


Assignees

mernst

Filing an Issue

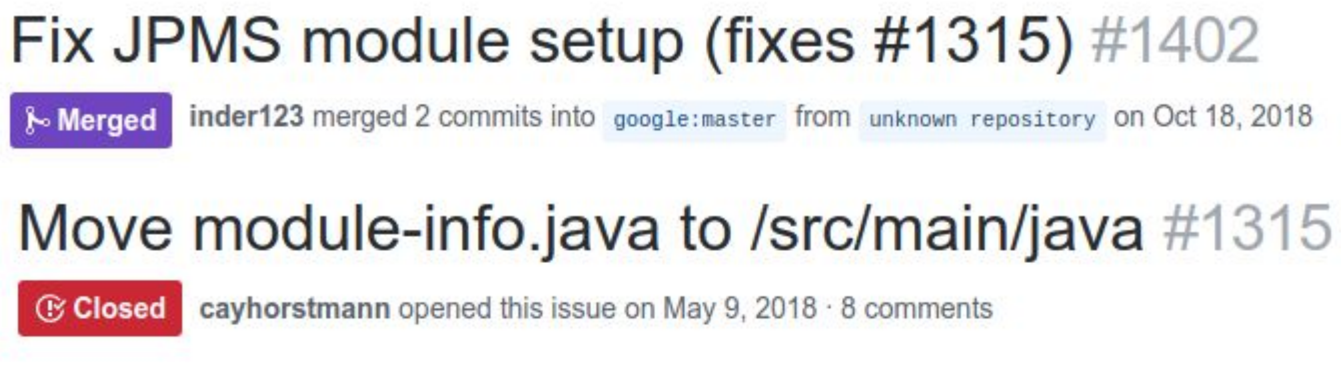
- Refer to report IDs in commit messages and pull requests addressing such issues.
- Allows traceability, showing progress in addressing issues.



Fix JPMS module setup (fixes #1315) (#1402)

```
* Fix JPMS module setup (fixes #1315)

* Re-added cause to AssertionErrors
```



Fix JPMS module setup (fixes #1315) #1402

inder123 merged 2 commits into `google:master` from `unknown repository` on Oct 18, 2018

Move module-info.java to /src/main/java #1315

cayhorstmann opened this issue on May 9, 2018 · 8 comments

Issue Reporting Advice

- Only include one bug per issue.
  - If an issue covers multiple bugs, it is harder to track progress and ensure all bugs are fixed.
  - Allows flexible assigning of priority and responsibility.
  - If existing issues cover multiple bugs, split them into unique issues.
    - If linked, refer to their issue IDs in each report.
- Use descriptive subjects.
  - Issue title is used to get an idea of the contents at a glance. Very important to convey what exactly is wrong in the title.
  - Tracker might have hundreds of issues. Important to let developers search information efficiently

Issue Reporting Advice

- Focus on the facts.
    - Do not speculate on the cause of an issue if you can avoid it. Give exact facts rather than guessing.
    - If you have an educated guess, you can provide it. Just be careful, as these often are wrong.
- Describe your expectations.
    - Describe what you expected to happen when you used the system. Some issues are subjective.
    - Describe what actually happened. Include output text, a screenshot, a URL if you can.
        - “I can’t log in” versus “I’m at (URL) and I’m trying to log in using my username, (username). But the system is saying that my username can’t be found, and it’s taking me to (URL).”

Issue Reporting Advice

- Document each problem as soon as you encounter it.
  - Then try to reproduce the problem based on the steps you just wrote down.
- Even if you can’t reproduce the problem, it’s still worth reporting.
  - Sometimes multiple reports of an issue are the only way a developer can track it down.
- If you can reproduce, try to reduce the sequence of steps down to the simplest sequence that still triggers the issue.

Issue Reporting Advice

- Before reporting an issue, check whether it was already reported.
- Include platform information.
  - What OS are you using? Which version?
  - What hardware is in your machine?
- Describe the severity of the issue.
  - **Blocker:** No further testing work can be done.
  - **Critical:** Application crash, Loss of data.
  - **Major:** Major loss of function.
  - **Minor:** Minor loss of function.
  - **Trivial:** Some UI enhancements.

Issue Reporting Advice

- Report issues immediately.
    - This ensures detailed and reproducible reports.
- Reproduce the bug three times before writing the report.
    - Make sure your steps are robust enough to reproduce the bug without ambiguity.
- See if the issue occurs in similar modules.
    - If code is repeated or used, see if you can corrupt the dependent modules.
    - May allow identification of most severe appearance of fault.

We Have Learned

- Your project will evolve over time. Tool support can ensure that this process is smooth.
- Version control can be used to control how code and other artifacts evolve.
    - A history of snapshots is maintained and can be revisited.
    - Mistakes can be easily resolved by rolling back to a snapshot.
    - Allows multiple developers to collaborate.
        - Branches and forks allow experimental development while maintaining a stable version.

We Have Learned

- Issue trackers allow developers and end-users to report bugs.
  - Reports should detail the user’s expectations, what actually happened, and steps or tests that allow reproduction of the issue.
  - System can be used to track whether issues are fixed, what issues affected the system in the past, and who is responsible for fixing issues.
  - Can also be used to manage feature requests and progress towards implementing them.
  - Issue IDs can be references in code commit messages and pull requests.

Next Time

- Testing Fundamentals
    - What is testing?
    - What are the stages of testing?
    - Sommerville, Chapter 8

- Homework 1
    - Due February 10th
    - **Ask questions online!**