Open Source Software Development Process

Yongsen MA

Shanghai Jiao Tong University

E-mail: mayongsen@gmail.com

I. Introduction

Aaron Koblin: Artfully visualizing our humanity

Why free software has poor usability, and how to improve it

A. *motivating, joining, participating and contributing*

1) acquire: knowledge, experience, opportunities; backup, platform
2) participate: happiness, communication
3) contribute: freedom, trustworthy

1) developer
2) user(evaluation)

1) public
2) private

B. *modeling, examination, investigation*

1) individuals
2) groups
3) organizations

1) operate systems
2) web
3) application
4) network

1) contribute:

2) process: stable, scalable
3) acquire: software, individuals, groups

1) graph theory
2) multiproject
3) interdependent

II. Open Source Project

A. Components

1) Home Page
2) Code Repository
3) Mailing List
4) Bug Tracking System
5) Wiki

B. Participating

1) Starting
2) Discussion
   - Subscribe Mailing List
   - Take part in News Group
   - Participate in Conference
3) **Programming and Debugging**
   - Consummate documents
   - Running test codes
   - Report Bugs
   - Submit patch
4) Improving

C. Developing

1) Creating a Repository
2) Making Changes

- Adding Files
- Committing Changes
- Files Status and Differences
- Managing Files

3) **Managing Branches**

- Creating Branches
- Merging Branches
- Handling Conflicts
- Deleting and renaming branches

4) Handling Releases

III. Programming and Debugging

For example, how are crash reports handled? How are bug reports handled? How are bugs classified and confirmed?

A. Basic Debugging

B. Functional Debugging

IV. Maintaining and Branching

How are the assignments to individual developers made? How to merge code changes in Git? How are code inconsistency handled? In each step of the process, have you identified any software engineering issues which have rooms for improvements?

A. Branching

Branching means you diverge from the main line of development and continue to do work without messing with that main line. The way Git branches is incredibly lightweight, making branching operations nearly instantaneous and switching back and forth between branches generally just as fast.

A branch in Git is simply a lightweight movable pointer to one of these commits. The default branch name in Git is master. As you initially make commits, youre given a master branch that points to the last commit you made. Every time you commit, it moves forward automatically.

1) **Creating Branching:** To create a branch and switch to it at the same time, you can run:

   ```
   git checkout -b example
   ```

   Then you can do the following work in the branch:

   - Test Changing
   - Add new functionality
   - Fix bugs

   At this stage, youll receive a call that another issue is critical and you need a hotfix. Youll do the following:

   a) Revert back to your production branch.
   b) Create a branch to add the hotfix.
   c) After its tested, merge the hotfix branch, and push to production.
   d) Switch back to your original story and continue working.

2) **Merging Branching:** Combine directory and file contents from separate sources to yield one combined result. Suppose youve decided that your issue is complete and ready to be merged into your master branch. All you have to do is check out the branch you wish to merge into and then run the git merge command:

   ```
   git checkout master
   git merge example
   ```

   - Sources for merges are local branches
   - Merges always occur in the current, checked-out branch
   - A complete merge ends with a new commit

   Merge heuristics:

   - Several merge strategies: resolve, recursive, octopus, ours
   - Techniques: fastforward, threeway

   Merge types

   - straight merge
   - squashed commits
   - cherry picking

3) **Handling Conflicts:** Occasionally, this process doesnt go smoothly. If you changed the same part of the same file differently in the two branches youre merging together, Git

wont be able to merge them cleanly. Youll get a merge conflict information. It has paused the process while you resolve the conflict. If you want to see which files are unmerged at any point after a merge conflict, you can run git status:

```
git status
```

You can handle this conflict by manual or other tools.

a) Manual: Git adds standard conflict-resolution markers to the files that have conflicts, so you can open them manually and resolve those conflicts.
b) Tools: If you want to use a graphical tool to resolve these issues, you can run git mergetool, which fires up an appropriate visual merge tool and walks you through the conflicts:

```
git mergetool
```

4) **Deleting and renaming branches:** Now that your work is merged in, you have no further need for the example branch. You can delete it and then manually close the ticket in your tracking system:

```
git branch -d example
```

B. Other Issues

1) **Pushing:** When you want to share a branch with the world, you need to push it up to a remote that you have write access to. Your local branches arent automatically synchronized to the remotes you write to you have to explicitly push the branches you want to share. That way, you can use private branches for work you dont want to share, and push up only the topic branches you want to collaborate on.
2) **Tracking:** Checking out a local branch from a remote branch automatically creates what is called a tracking branch. Tracking branches are local branches that have a direct relationship to a remote branch. If youre on a tracking branch and type git push, Git automatically knows which server and branch to push to. Also, running git pull while on one of these branches fetches all the remote references and then automatically merges in the corresponding remote branch.

3) **Rebasing:** In Git, there are two main ways to integrate changes from one branch into another: the merge and the rebase.

C. Sending Changes Upstream

- Generate and send patches via email
  - Most developers send patches to a maintainer or list
  - Highly visible public review of patches on mail list
- Maintainer pulls updates from a downstream developer
  - Maintainer can directly pull from your published repository
  - Initiated by upstream maintainer
- Developer pushes updates to an upstream maintainer
  - Some developers have write permissions on an upstream repository
  - Initiated by downstream developer

REFERENCES

[1] A. Bonaccorsi and C. Rossi. Why open source software can succeed. *Research Policy*, 32(7):1243 – 1258, 2003.

[2] S. Chacon, J. Hamano, and S. Pearce. *Pro Git*, volume 288. Apress, 2009.

[3] G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy*, 32(7):1159 – 1177, 2003.

[4] B. Kernighan and R. Pike. *The practice of programming*. Addison-Wesley Professional, 1999.

[5] B. Kogut and A. Metiu. Opensource software development and distributed innovation. *Oxford Review of Economic Policy*, 17(2):248–264, 2001.

[6] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani. Understanding free/open source software development processes. *Software Process: Improvement and Practice*, 11(2):95–105, 2006.

[7] G. von Krogh and E. von Hippel. Special issue on open source software development. *Research Policy*, 32(7):1149 – 1157, 2003.

[8] C. Yilmaz, A. M. Memon, A. Porter, A. S. Krishna, D. C. Schmidt, and A. Gokhale. Techniques and processes for improving the quality and performance of open-source software. In *Software Process: Improvement and Practice*. John Wiley & Sons, 2006.