“Merge and Graft: Two Twins That Need To Grow Apart”

Robin La Fontaine

Nigel Whitaker

DeltaXML

Merge is important in version control systems, e.g. Git

- Merge conflicts take time and effort to sort out

- XML/JSON aware merge is better than line-based merge

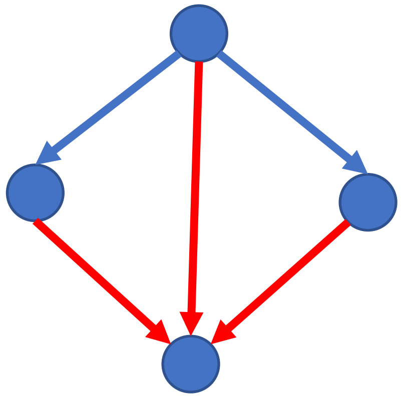
- XML and XPath/XSLT allow rules to be applied
  - Enables us to have different types of merge
  - Some conflicts can be avoided
  - Some conflicts can be resolved automatically

Merge and Graft (Cherry-pick) in Git

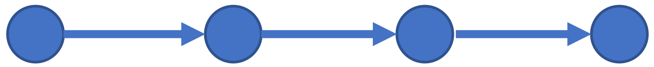
- Our objective: make life easier for anyone merging XML or JSON in Git
  - Improved merge/graft tools
  - Fewer conflicts to resolve manually (takes time and is tedious)

- Our approach: Provide XML and JSON aware merge and graft tools
  - We will show Merge and Graft are not the same
  - Rule-based merge/graft can help
  - Integration into Git is work in progress

“Varieties of XML Merge: Concurrent versus Sequential”, presented at XML Prague 2018

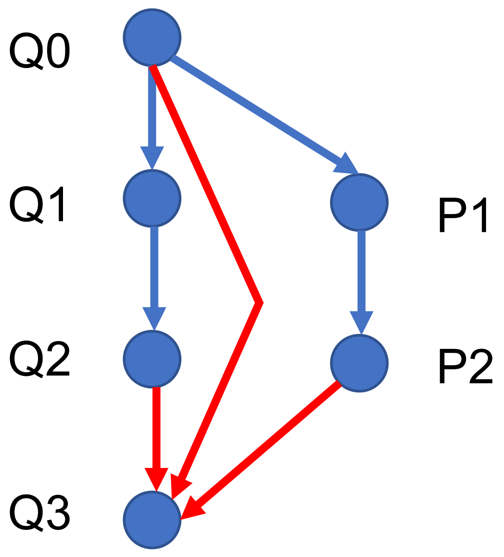


Concurrent Merge

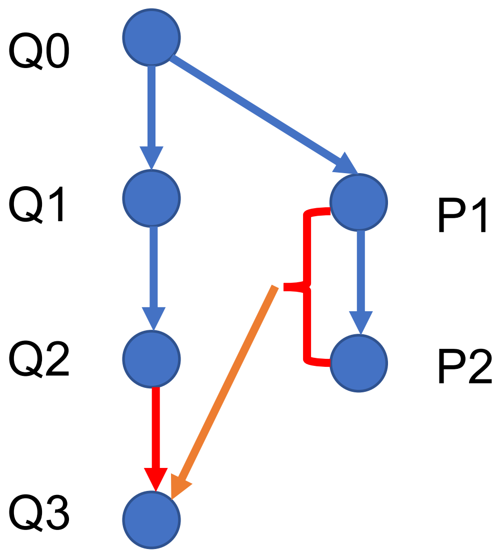


Sequential Merge

Merge and Graft (Cherry-pick)

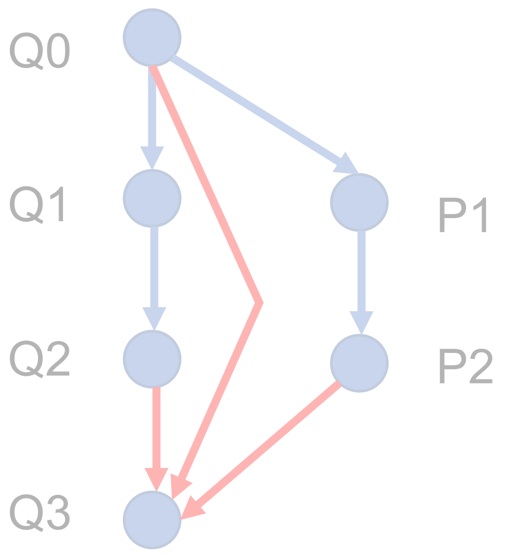


Merge
Q2+P2 to
create Q3

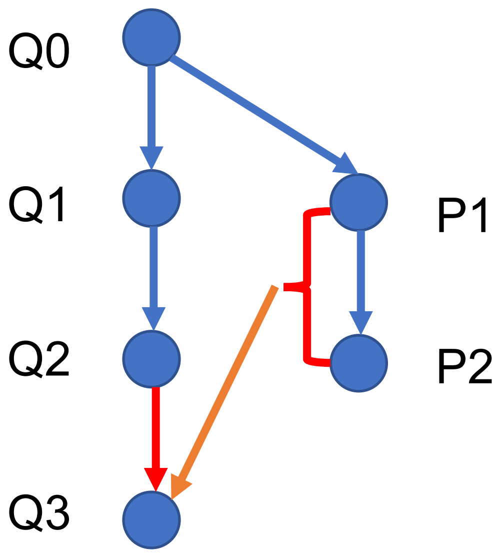


Graft P1->P2
changes to create
Q3

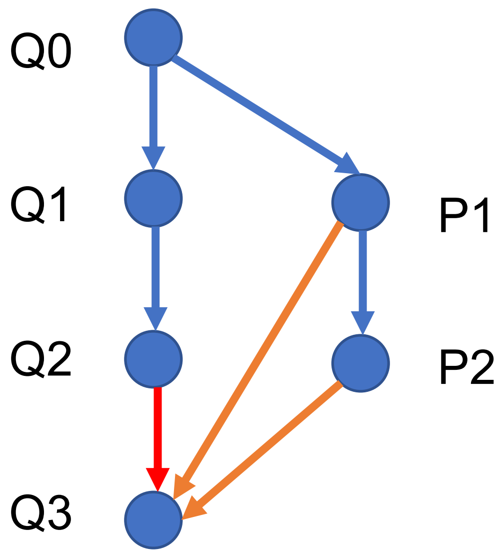
Are merge and graft the same?



Merge Q2+P2 to create Q3



Graft P1->P2 changes to create Q3



Implementing graft as a merge

DELTAXML

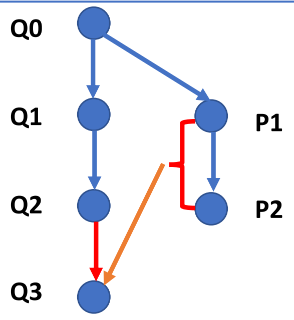
Graft: apply changes P1->P2 to Q2

```
P1
{
   "John": "v2",
   "Mike": "v1",
   "Anna": "v1",
   "David": "v1"

}
```

```
P2
{

   "Mike": "v2",
   "Anna": "v1",
   "David": "v2"

}
```

```
Q2
{
   "John": "v1",
   "Mike": "v1",
   "Anna": "v2",

   "Jane": "v1"
}
```

```
Q3: Graft
{

   "Mike": "v2",
   "Anna": "v2",

   "Jane": "v1"
}
```



Graft P1->P2 changes to create Q3

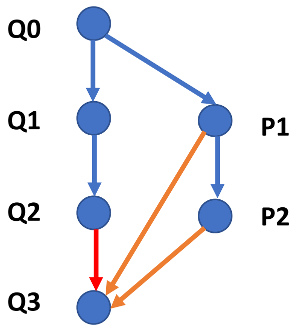
Merge: merge changes in P2 and Q2

```
P1
{
   "John": "v2",
   "Mike": "v1",
   "Anna": "v1",
   "David": "v1"

}
```

```
P2
{

   "Mike": "v2",
   "Anna": "v1",
   "David": "v2"

}
```

```
Q2
{
   "John": "v1",
   "Mike": "v1",
   "Anna": "v2",

   "Jane": "v1"
}
```

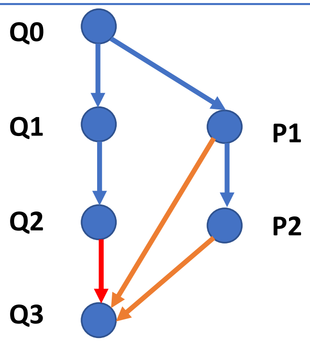
```
Q3: Merge
{
   !CONFLICT
   "Mike": "v2",
   "Anna": "v2",
   !CONFLICT
   "Jane": "v1"
}
```



Implementing graft as a merge

Do we get Graft if we merge with Q priority?

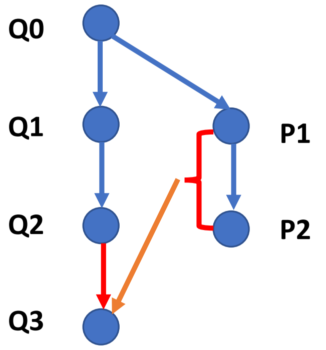
```
P1                  P2                  Q2                  Q3: Merge           Q3: Q2 Priority
{                   {                   {                   {                   {
 "John": "v2",                           "John": "v1",       !CONFLICT           "John": "v1",
 "Mike": "v1",       "Mike": "v2",       "Mike": "v1",       "Mike": "v2",       "Mike": "v2",
 "Anna": "v1",       "Anna": "v1",       "Anna": "v2",       "Anna": "v2",       "Anna": "v2",
 "David": "v1"       "David": "v2"                           !CONFLICT
                                         "Jane": "v1"        "Jane": "v1"        "Jane": "v1"
}                   }                   }                   }                   }
```



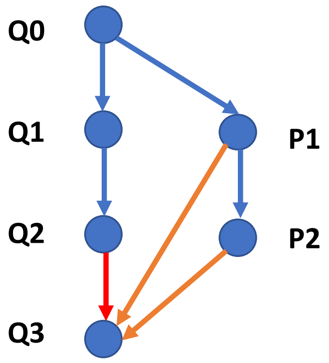
Implementing graft as a merge

Do we get Graft if we merge with Q priority? No!

```
Q3: Graft                Q3: Merge          Q3: Q2 Priority
{                      {                  {
                        !CONFLICT          "John": "v1",
 "Mike": "v2",          "Mike": "v2",      "Mike": "v2",
 "Anna": "v2",          "Anna": "v2",      "Anna": "v2",
                        !CONFLICT
 "Jane": "v1"           "Jane": "v1"       "Jane": "v1"
}                      }                   }
```



Graft P1->P2 changes to create Q3

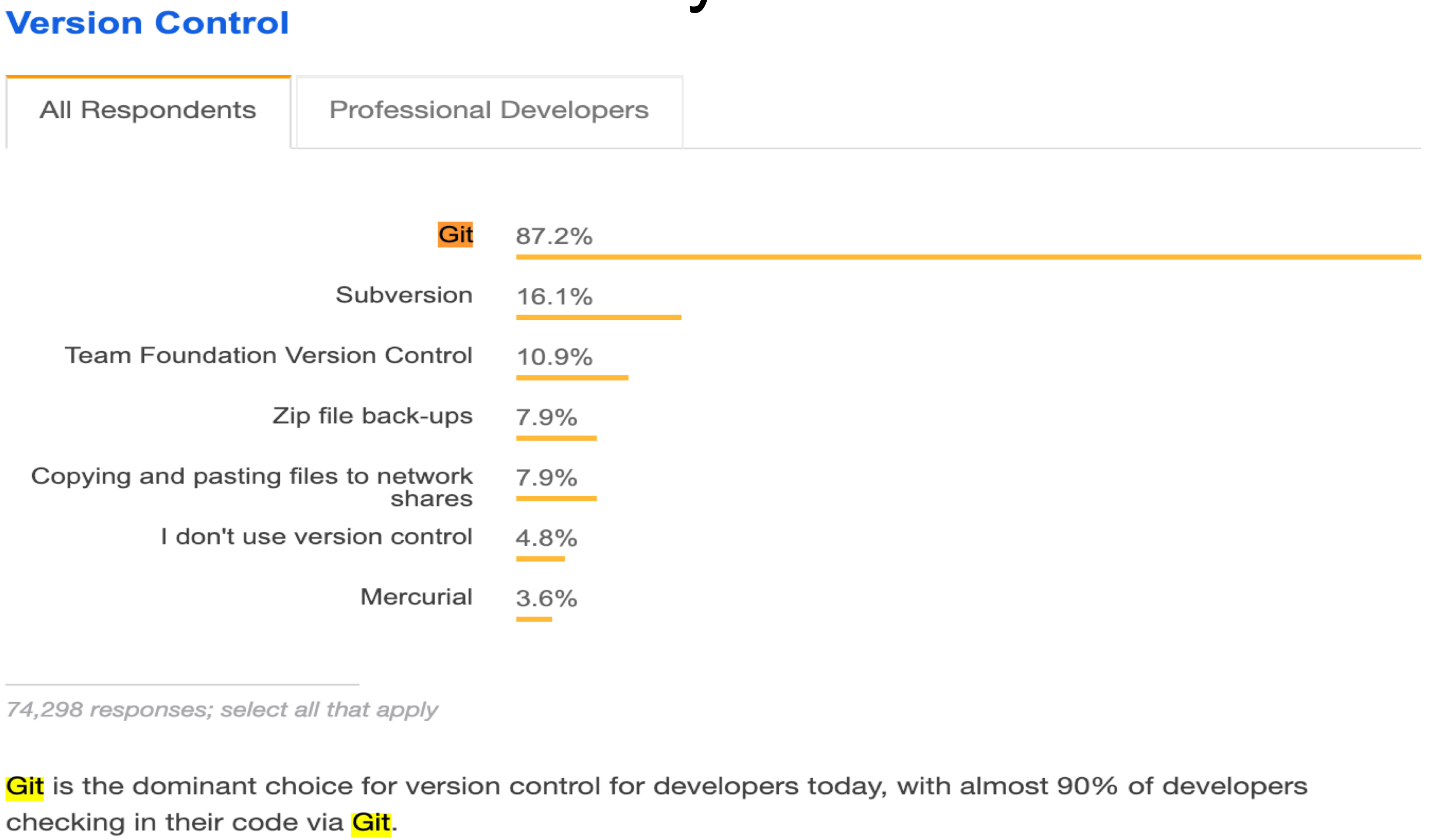


Implementing graft as a merge

The story so far…

- XML and JSON aware merge tools can give better results than line-based merge
- We have shown Merge and Graft are not the same
- BUT we will see that Git does not make this distinction
- So how does Git handle merge and is there scope to improve it?

Why Git?



Version Control

Git	87.2%
Subversion	16.1%
Team Foundation Version Control	10.9%
Zip file back-ups	7.9%
Copying and pasting files to network shares	7.9%
I don't use version control	4.8%
Mercurial	3.6%

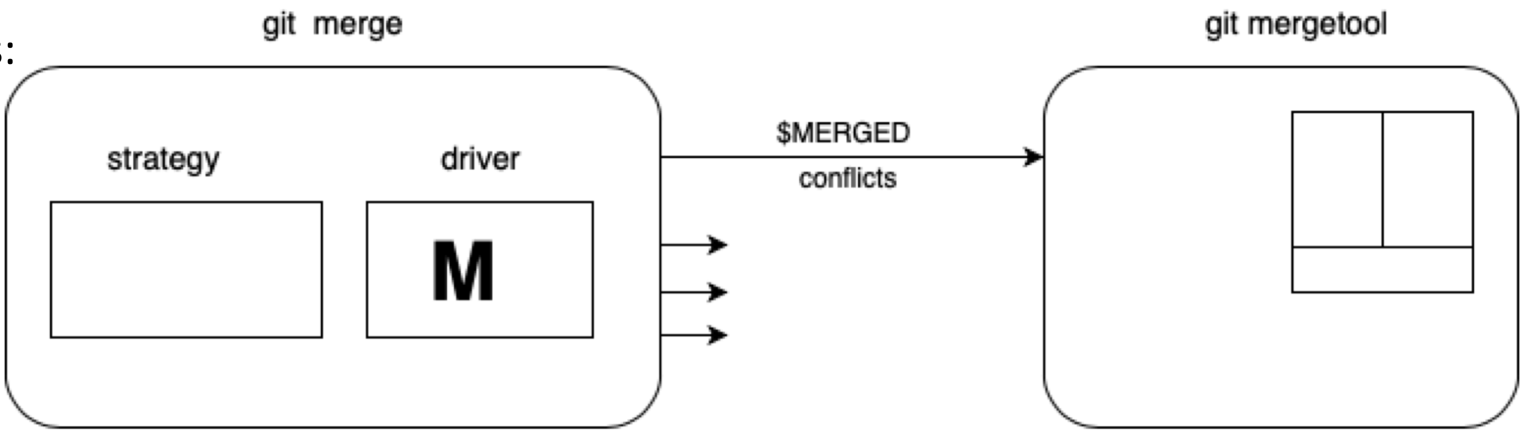
74,298 responses; select all that apply

Git is the dominant choice for version control for developers today, with almost 90% of developers checking in their code via Git.

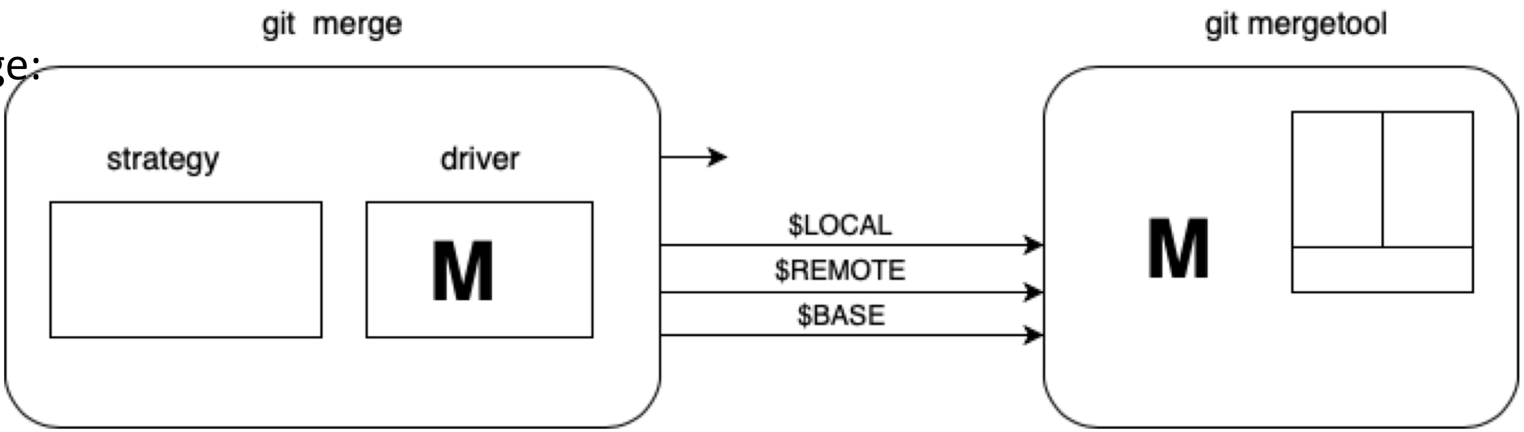
Source: StackOverflow developer survey 2018: https://insights.stackoverflow.com/survey/2018/

Git merge workflows

Passing conflicts:



Re-merge:



Merge conflict discrepancies

```
<p id='conclusions'>All is well and good!</p>
```

```
<p id='conclusions' xml:lang="en_GB">All is well and good!</p>
```

```
<p xml:lang="en_GB" id='conclusions' >All is well and good!</p>
```

```
<<<<<<< A
    <p id='conclusions' xml:lang="en_GB">All is well and good!</p>
=======
    <p xml:lang="en_GB" id='conclusions' >All is well and good!</p>
>>>>>>> B
```

Non-conflicting, bad text merge

```
<rule-set name="Incoming Public”
  target-interface="PublicLAN”
  no-match-action="drop">
  <rule name="allow https for website failover”
    target-ip="81.2.96.130”
    target-port="443”
    action="accept"/>
</rule-set>
```

```
<rule-set name="Incoming Public”
  target-interface="PublicLAN”
  no-match-action="drop">
  <rule name="allow https for website failover”
    target-ip="81.2.96.130”
    target-port="443”
    protocol="6”
    action="accept"/>
</rule-set>
```

```
<rule-set name="Incoming Public”
  target-interface="PublicLAN”
  no-match-action="drop">
  <rule name="allow https for website failover’
    protocol="6”
    target-ip="81.2.96.130”
    target-port="443”
    action="accept"/>
</rule-set>
```

```
<rule-set name="Incoming Public”
  target-interface="PublicLAN”
  no-match-action="drop">
  <rule name="allow https for website failover”
    protocol="6”
    target-ip="81.2.96.130”
    target-port="443”
    protocol="6”
    action="accept"/>
</rule-set>
```

Merge Driver Setup

Download the repo onto your file system. Note the path to the bin folder.

Create .gitattributes with patterns in your git repository to associate json or xml files with the merge drivers. For example:

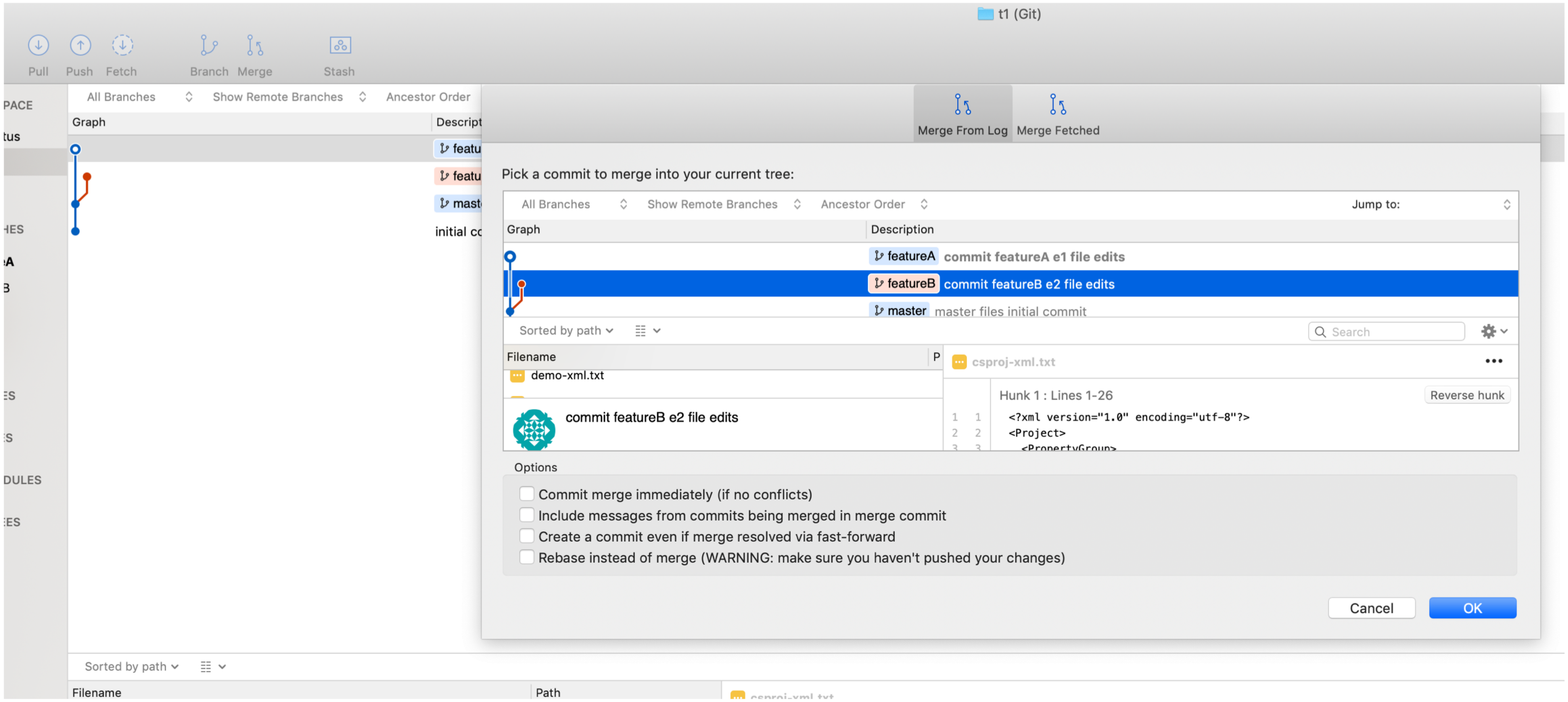
```
*.xml  merge=xmlmerge
*.json merge=jsonmerge
```

Then in git config configure the xml and json merge drivers, using --local, --global or --system as appropriate:

```
$ git config --local merge.xmlmerge.name "DeltaXML XML Merge"
$ git config --local merge.xmlmerge.driver "/Users/nigelw/bin/git-xml-merge-driver %O %A %B %L %P"
$ git config --local merge.jsonmerge.name "DeltaXML JSON Merge"
$ git config --local merge.jsonmerge.driver "/Users/nigelw/bin/git-json-merge-driver %O %A %B %L %P"
```

> “Note: The path to the drivers must be an absolute filesystem path and correspond to the location where you saved the files in the bin folder.
> ”

Merge workflow (passing conflicts)



Pick a commit to merge into your current tree:

Graph	Description
	featureA commit featureA e1 file edits
	featureB commit featureB e2 file edits
	master master files initial commit

demo-xml.txt

commit featureB e2 file edits

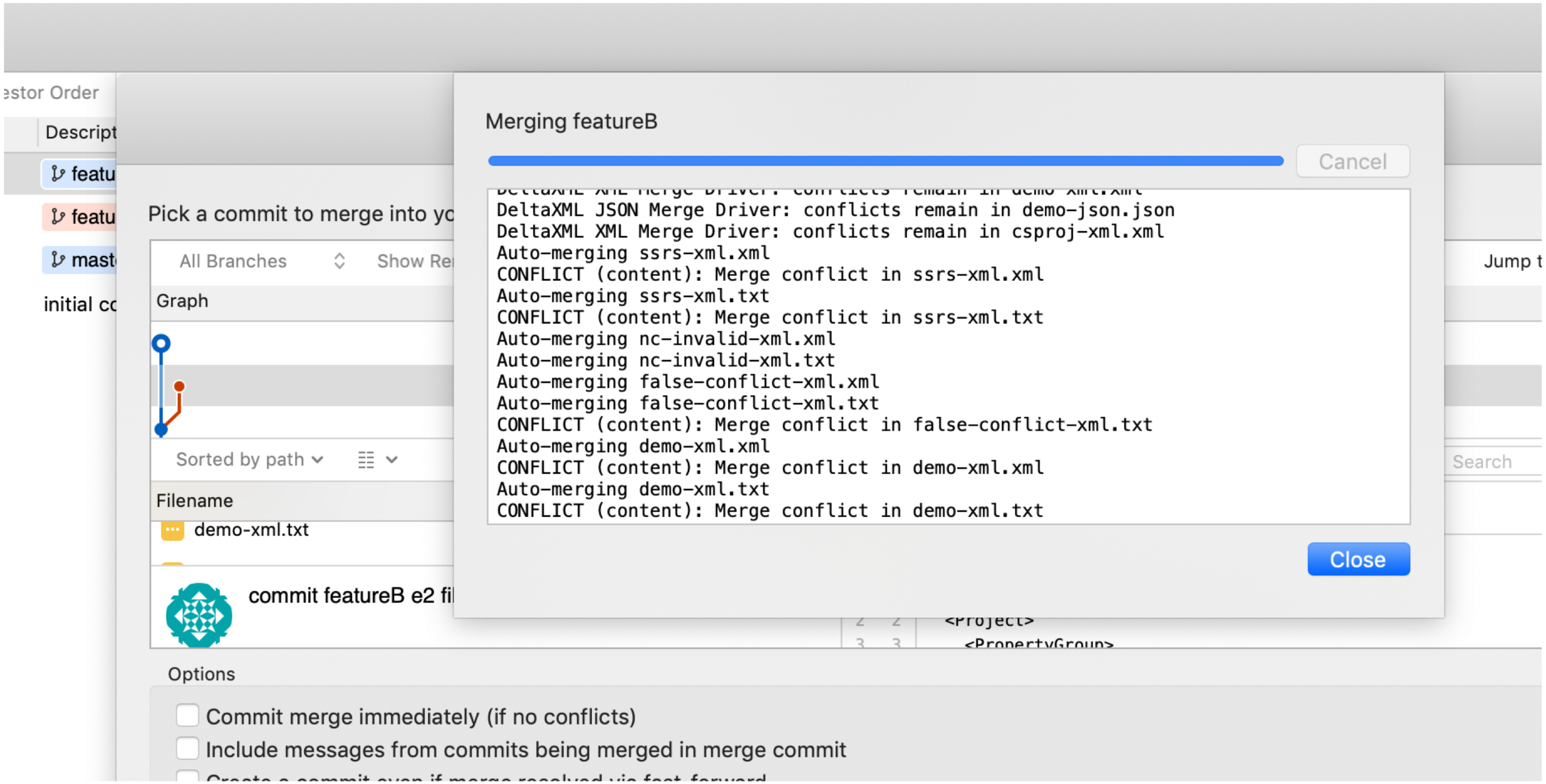
csproj-xml.txt

Hunk 1 : Lines 1-26

```
<?xml version="1.0" encoding="utf-8"?>
<Project>
  <PropertyGroup>
```

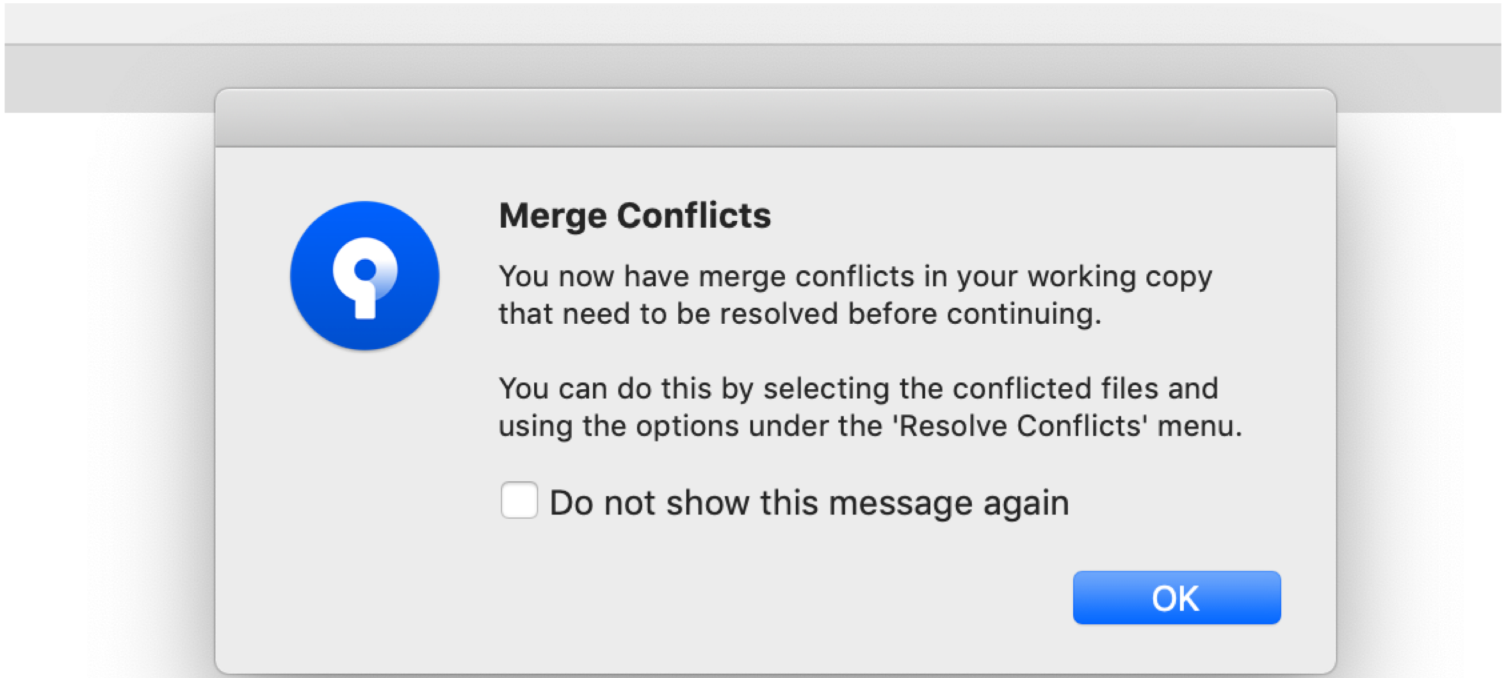
Options

- [ ] Commit merge immediately (if no conflicts)
- [ ] Include messages from commits being merged in merge commit
- [ ] Create a commit even if merge resolved via fast-forward
- [ ] Rebase instead of merge (WARNING: make sure you haven't pushed your changes)



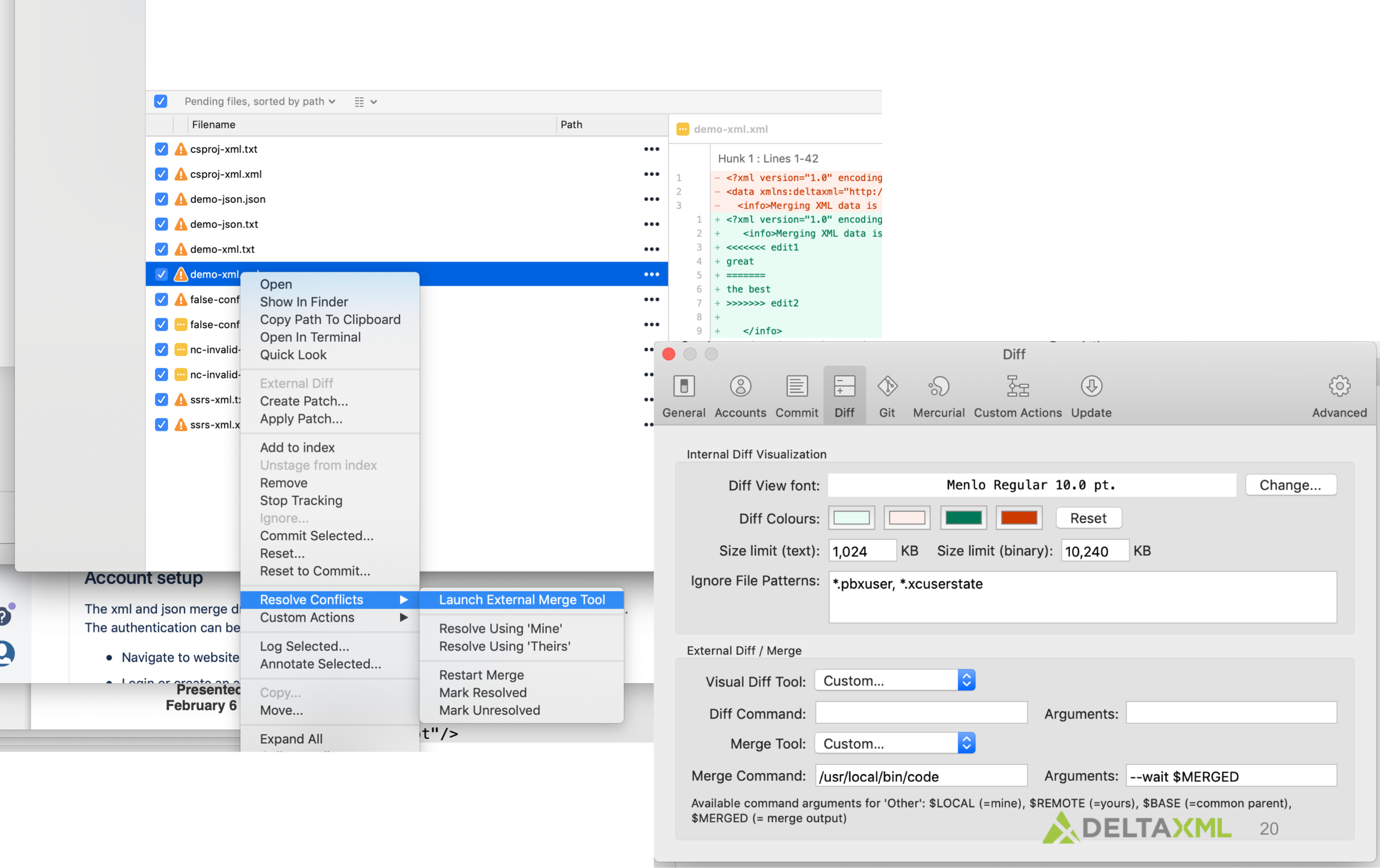
```
DeltaXML JSON Merge Driver: conflicts remain in demo-json.json
DeltaXML XML Merge Driver: conflicts remain in csproj-xml.xml
Auto-merging ssrs-xml.xml
CONFLICT (content): Merge conflict in ssrs-xml.xml
Auto-merging ssrs-xml.txt
CONFLICT (content): Merge conflict in ssrs-xml.txt
Auto-merging nc-invalid-xml.xml
Auto-merging nc-invalid-xml.txt
Auto-merging false-conflict-xml.xml
Auto-merging false-conflict-xml.txt
CONFLICT (content): Merge conflict in false-conflict-xml.txt
Auto-merging demo-xml.xml
CONFLICT (content): Merge conflict in demo-xml.xml
Auto-merging demo-xml.txt
CONFLICT (content): Merge conflict in demo-xml.txt
```

- [ ] Commit merge immediately (if no conflicts)
- [ ] Include messages from commits being merged in merge commit


Merge Conflicts

You now have merge conflicts in your working copy that need to be resolved before continuing.

You can do this by selecting the conflicted files and using the options under the 'Resolve Conflicts' menu.



```
- <?xml version="1.0" encoding
- <data xmlns:deltaxml="http:/
-   <info>Merging XML data is
+ <?xml version="1.0" encoding
+    <info>Merging XML data is
+ <<<<<<< edit1
+ great
+ =======
+ the best
+ >>>>>>> edit2
+
+    </info>
```

Internal Diff Visualization

Diff View font: Menlo Regular 10.0 pt.

Diff Colours: ▭ ▭ ■ ■

Size limit (text): 1,024 KB Size limit (binary): 10,240 KB

Ignore File Patterns: *.pbxuser, *.xcuserstate

External Diff / Merge

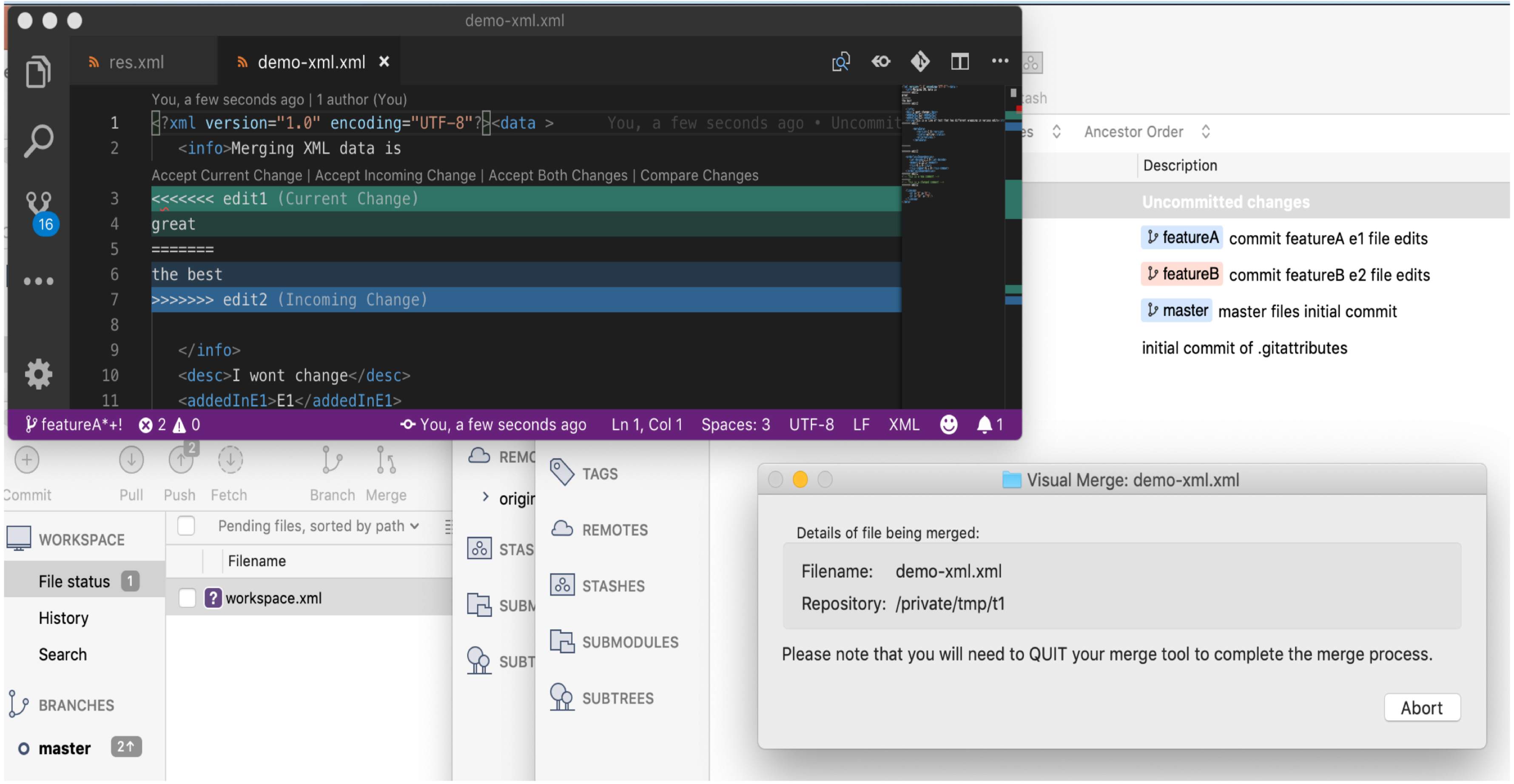
Visual Diff Tool: Custom...

Diff Command: Arguments:

Merge Tool: Custom...

Merge Command: /usr/local/bin/code Arguments: --wait $MERGED

Available command arguments for 'Other': $LOCAL (=mine), $REMOTE (=yours), $BASE (=common parent), $MERGED (= merge output)



demo-xml.xml

res.xml | demo-xml.xml

You, a few seconds ago | 1 author (You)

```
<?xml version="1.0" encoding="UTF-8"?><data >     You, a few seconds ago • Uncommit
   <info>Merging XML data is
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< edit1 (Current Change)
great
=======
the best
>>>>>>> edit2 (Incoming Change)

   </info>
   <desc>I wont change</desc>
   <addedInE1>E1</addedInE1>
```

featureA*+! ⊗ 2 ⚠ 0 · You, a few seconds ago · Ln 1, Col 1 · Spaces: 3 · UTF-8 · LF · XML · 1

Ancestor Order

Description
Uncommitted changes
featureA commit featureA e1 file edits
featureB commit featureB e2 file edits
master master files initial commit
initial commit of .gitattributes

Pending files, sorted by path

Filename
workspace.xml

Visual Merge: demo-xml.xml

Details of file being merged:

Filename: demo-xml.xml
Repository: /private/tmp/t1

Please note that you will need to QUIT your merge tool to complete the merge process.

Abort

DELTAXML

Conclusions

- XML and JSON aware merge tools can give better results than line-based merge
  - Fewer conflicts
  - Best done in Git Merge Driver

- Merge and Graft (cherry-pick) are arguably not the same
  - But Merge and Graft are treated the same way in Git

- Communication of conflicts from Merge Driver to Merge Tool needs to be improved
  - To handle conflicts in tree-structured data/documents