The Sector Specialists

Source control & branching

Submitted by: Grzegorz Jachimko
on: November 2014
Version: 1.0

Confidential

Agenda

Why source control in the first place?

The 10 golden rules

Next level → branches

Exercise

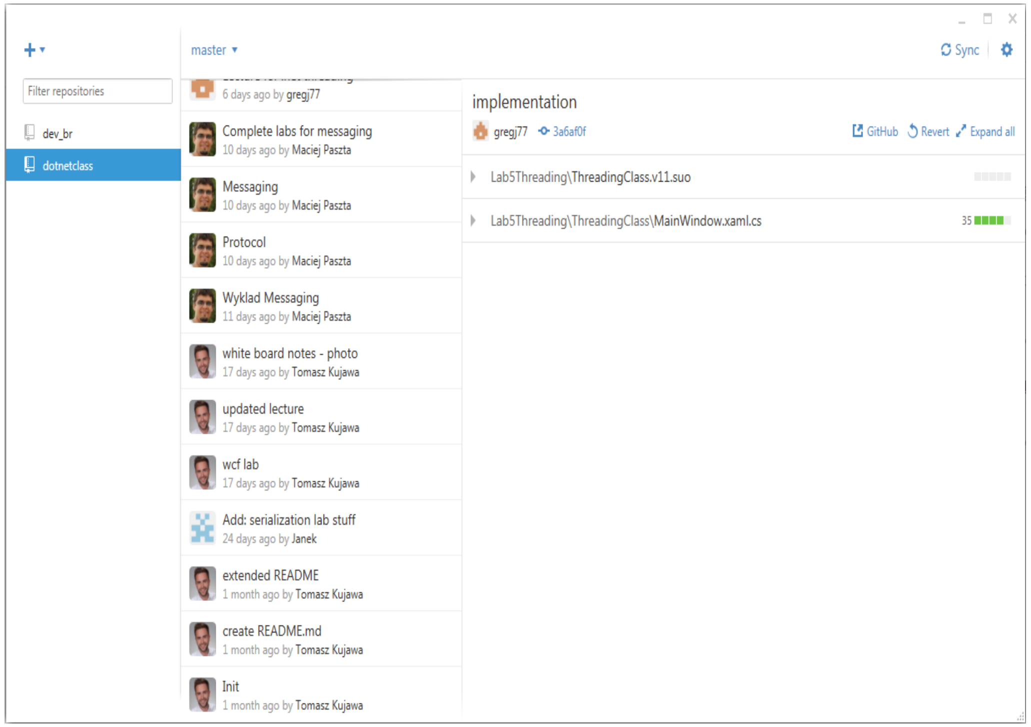
Why bother with source control?

- There is no straight answer, but let me ask you couple of questions:
  - Have you ever made a change, which caused application to stop working completely, and you want to take a step back?
  - Have you lost code (due to HW or SW failure)?
  - … and had a chance to restore it from stale backup?
  - Had multiple versions of the software?
  - Wanted to compare these two versions?
  - Wanted to review changes hisotry?
  - Wanted to change somebody’s else code?
  - Had to work in a team?
  - See who made a change?
  - Wanted to experiment with new feature or approach?

The 10 golden rules

1. Use it!

- Lot of different options available
- It should be a basic rule for every project!



6 days ago by gregj77

Complete labs for messaging
10 days ago by Maciej Paszta

Messaging
10 days ago by Maciej Paszta

Protocol
10 days ago by Maciej Paszta

Wyklad Messaging
11 days ago by Maciej Paszta

white board notes - photo
17 days ago by Tomasz Kujawa

updated lecture
17 days ago by Tomasz Kujawa

wcf lab
17 days ago by Tomasz Kujawa

Add: serialization lab stuff
24 days ago by Janek

extended README
1 month ago by Tomasz Kujawa

create README.md
1 month ago by Tomasz Kujawa

Init
1 month ago by Tomasz Kujawa

implementation
gregj77 3a6af0f

Lab5Threading\ThreadingClass.v11.suo
Lab5Threading\ThreadingClass\MainWindow.xaml.cs 35

The 10 golden rules

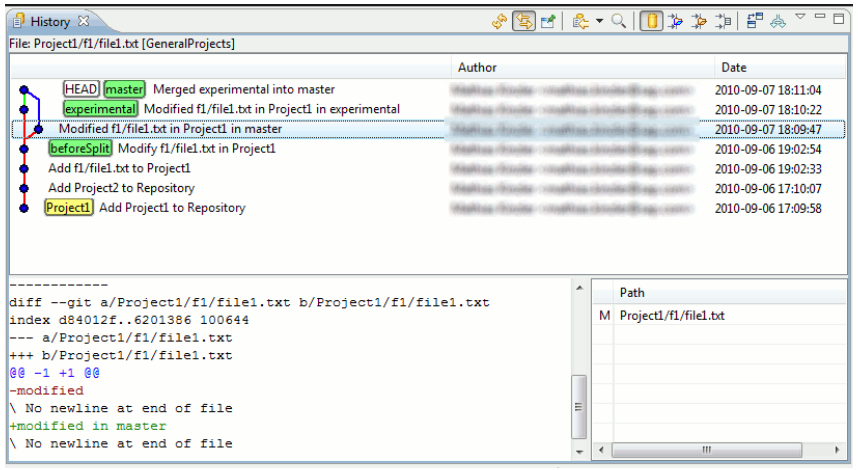
2. If it’s not in source control – it doesn’t exist

- Only code in the repo can be measured
- Backup possibility
- Tracking

The 10 golden rules

3. Commit often

- Checkpoints
- Merge nightmare
- Managable units of work



History

File: Project1/f1/file1.txt [GeneralProjects]

	Author	Date
HEAD master Merged experimental into master	[redacted]	2010-09-07 18:11:04
experimental Modified f1/file1.txt in Project1 in experimental	[redacted]	2010-09-07 18:10:22
Modified f1/file1.txt in Project1 in master	[redacted]	2010-09-07 18:09:47
beforeSplit Modify f1/file1.txt in Project1	[redacted]	2010-09-06 19:02:54
Add f1/file1.txt to Project1	[redacted]	2010-09-06 19:02:33
Add Project2 to Repository	[redacted]	2010-09-06 17:10:07
Project1 Add Project1 to Repository	[redacted]	2010-09-06 17:09:58

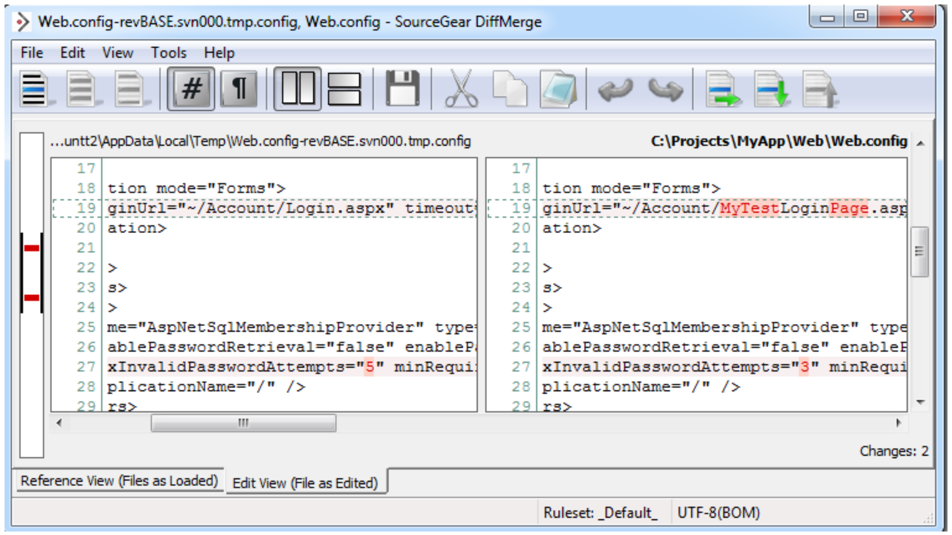
```
------------
diff --git a/Project1/f1/file1.txt b/Project1/f1/file1.txt
index d84012f..6201386 100644
--- a/Project1/f1/file1.txt
+++ b/Project1/f1/file1.txt
@@ -1 +1 @@
-modified
\ No newline at end of file
+modified in master
\ No newline at end of file
```

	Path
M	Project1/f1/file1.txt

The 10 golden rules

4. Review before commit

- Sanity check
- Personal or team **code review**
- Especially after merging!


Web.config-revBASE.svn000.tmp.config, Web.config - SourceGear DiffMerge
File Edit View Tools Help

...untt2\AppData\Local\Temp\Web.config-revBASE.svn000.tmp.config

```

tion mode="Forms">
ginUrl="~/Account/Login.aspx" timeout
ation>

>
s>
>
me="AspNetSqlMembershipProvider" type
ablePasswordRetrieval="false" enableP
xInvalidPasswordAttempts="5" minRequi
plicationName="/" />
rs>
```

C:\Projects\MyApp\Web\Web.config

```

tion mode="Forms">
ginUrl="~/Account/MyTestLoginPage.asp
ation>

>
s>
>
me="AspNetSqlMembershipProvider" type
ablePasswordRetrieval="false" enableP
xInvalidPasswordAttempts="3" minRequi
plicationName="/" />
rs>
```

Changes: 2

Reference View (Files as Loaded)
Edit View (File as Edited)
Ruleset: _Default_ UTF-8(BOM)

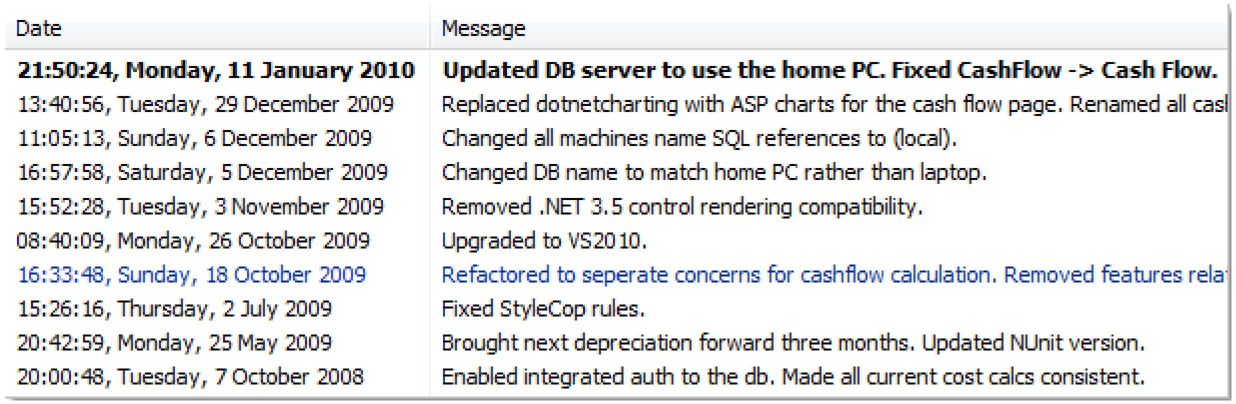
© Rule Financial 2013

7

The 10 golden rules

5. Proper commit messages

- Explains the reason behind commit
- Helps track the changes and progress
- Can link to requirements (JIRA)
- Do not use:
  - „fix", „refactor", „type-o fix", etc.

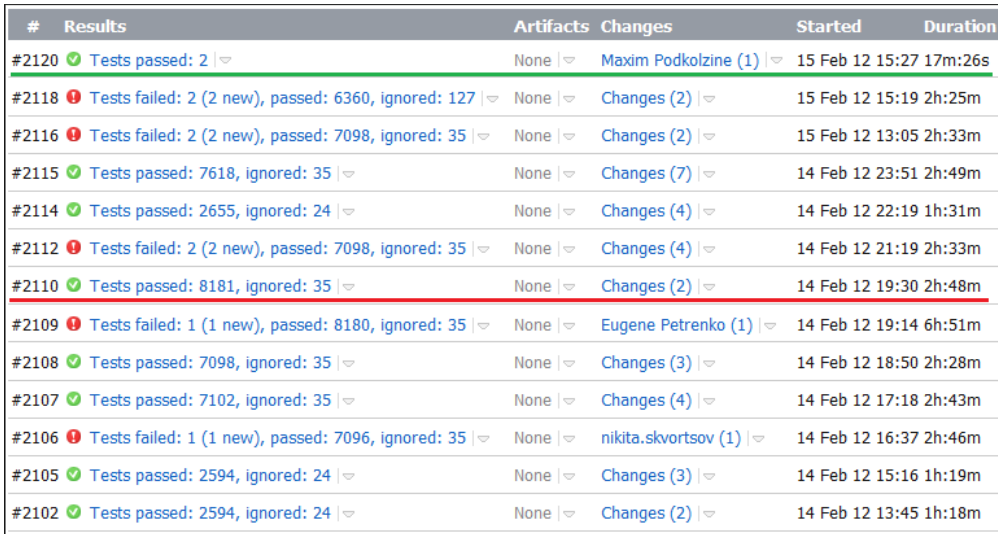


Date	Message
21:50:24, Monday, 11 January 2010	**Updated DB server to use the home PC. Fixed CashFlow -> Cash Flow.**
13:40:56, Tuesday, 29 December 2009	Replaced dotnetcharting with ASP charts for the cash flow page. Renamed all casl
11:05:13, Sunday, 6 December 2009	Changed all machines name SQL references to (local).
16:57:58, Saturday, 5 December 2009	Changed DB name to match home PC rather than laptop.
15:52:28, Tuesday, 3 November 2009	Removed .NET 3.5 control rendering compatibility.
08:40:09, Monday, 26 October 2009	Upgraded to VS2010.
16:33:48, Sunday, 18 October 2009	Refactored to seperate concerns for cashflow calculation. Removed features rela
15:26:16, Thursday, 2 July 2009	Fixed StyleCop rules.
20:42:59, Monday, 25 May 2009	Brought next depreciation forward three months. Updated NUnit version.
20:00:48, Tuesday, 7 October 2008	Enabled integrated auth to the db. Made all current cost calcs consistent.

The 10 golden rules

6. Commit your changes; never the last item in a day

- Only you have proper context!
- If you submit – make sure it works
- If you broke it – you fix it!
- If you broke it and left – you are an **a#$%^&**!

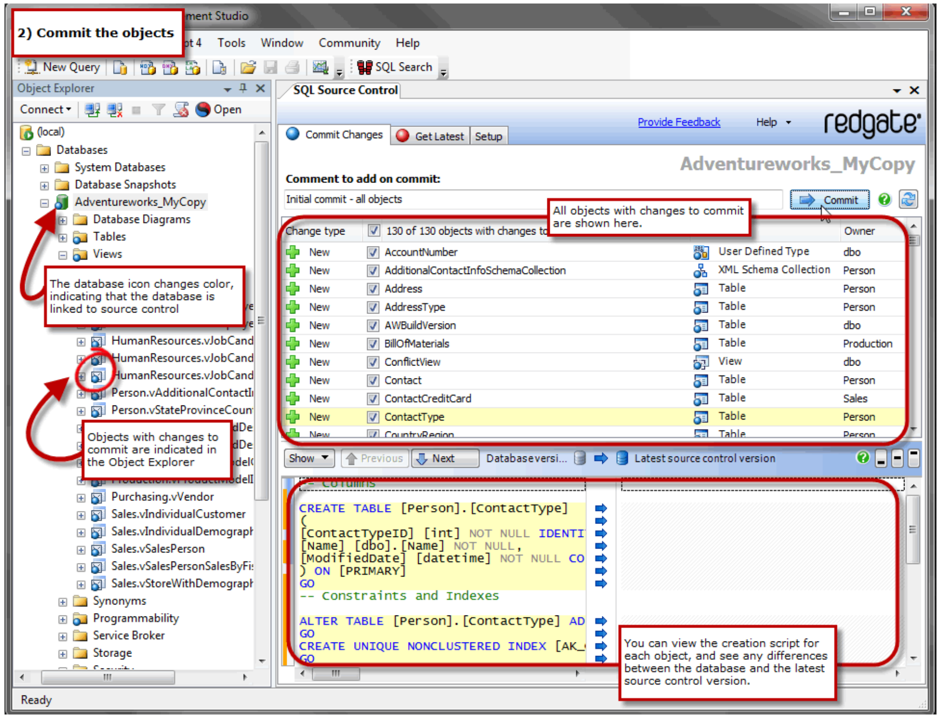


#	Results	Artifacts	Changes	Started	Duration
#2120	✓ Tests passed: 2	None	Maxim Podkolzine (1)	15 Feb 12 15:27	17m:26s
#2118	❗ Tests failed: 2 (2 new), passed: 6360, ignored: 127	None	Changes (2)	15 Feb 12 15:19	2h:25m
#2116	❗ Tests failed: 2 (2 new), passed: 7098, ignored: 35	None	Changes (2)	15 Feb 12 13:05	2h:33m
#2115	✓ Tests passed: 7618, ignored: 35	None	Changes (7)	14 Feb 12 23:51	2h:49m
#2114	✓ Tests passed: 2655, ignored: 24	None	Changes (4)	14 Feb 12 22:19	1h:31m
#2112	❗ Tests failed: 2 (2 new), passed: 7098, ignored: 35	None	Changes (4)	14 Feb 12 21:19	2h:33m
#2110	✓ Tests passed: 8181, ignored: 35	None	Changes (2)	14 Feb 12 19:30	2h:48m
#2109	❗ Tests failed: 1 (1 new), passed: 8180, ignored: 35	None	Eugene Petrenko (1)	14 Feb 12 19:14	6h:51m
#2108	✓ Tests passed: 7098, ignored: 35	None	Changes (3)	14 Feb 12 18:50	2h:28m
#2107	✓ Tests passed: 7102, ignored: 35	None	Changes (4)	14 Feb 12 17:18	2h:43m
#2106	❗ Tests failed: 1 (1 new), passed: 7096, ignored: 35	None	nikita.skvortsov (1)	14 Feb 12 16:37	2h:46m
#2105	✓ Tests passed: 2594, ignored: 24	None	Changes (3)	14 Feb 12 15:16	1h:19m
#2102	✓ Tests passed: 2594, ignored: 24	None	Changes (2)	14 Feb 12 13:45	1h:18m

The 10 golden rules

7. Database schema and scripts is also a code!

- Define a base line file
- Use incremental changes



The database icon changes color, indicating that the database is linked to source control

Objects with changes to commit are indicated in the Object Explorer

Comment to add on commit: Initial commit - all objects

All objects with changes to commit are shown here.

Change type	130 of 130 objects with changes to		Owner
New	AccountNumber	User Defined Type	dbo
New	AdditionalContactInfoSchemaCollection	XML Schema Collection	Person
New	Address	Table	Person
New	AddressType	Table	Person
New	AWBuildVersion	Table	dbo
New	BillOfMaterials	Table	Production
New	ConflictView	View	dbo
New	Contact	Table	Person
New	ContactCreditCard	Table	Sales
New	ContactType	Table	Person
New	CountryRegion	Table	Person

```
-- Columns

CREATE TABLE [Person].[ContactType]
(
[ContactTypeID] [int] NOT NULL IDENTI
[Name] [dbo].[Name] NOT NULL,
[ModifiedDate] [datetime] NOT NULL CO
) ON [PRIMARY]
GO
-- Constraints and Indexes

ALTER TABLE [Person].[ContactType] AD
GO
CREATE UNIQUE NONCLUSTERED INDEX [AK_
GO
```

You can view the creation script for each object, and see any differences between the database and the latest source control version.

The 10 golden rules

8. Compilation output – no go!

- Too much space
- It is useless
- It may breake the build

The 10 golden rules

9. … and so are user / machine settings

- Some people may have different settings

The 10 golden rules

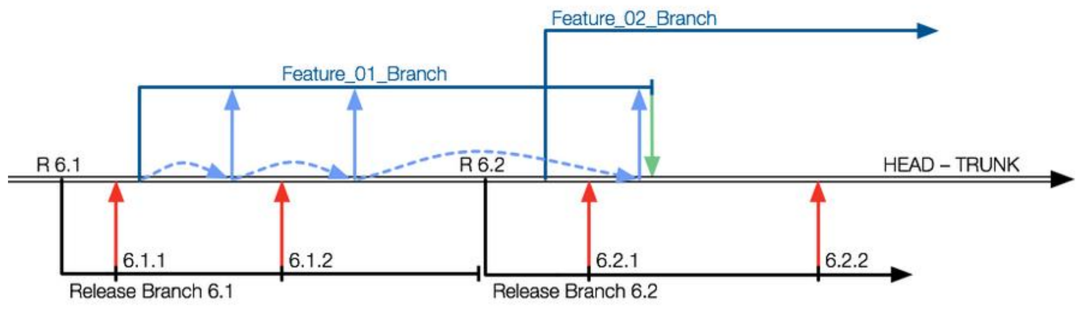
10. Don’t forget about dependencies!

- Small projects – define a dedicated LIB folder
- Large projects – consider nuget server

Next level - branches

Branches – what for?

- Define a solid and stable work environment – for developers and testers,
- Allows quick and continous releases from stable code base,
- Gives you chance to fix production issues,
- A tool for managing different project versions.



Next level - branches

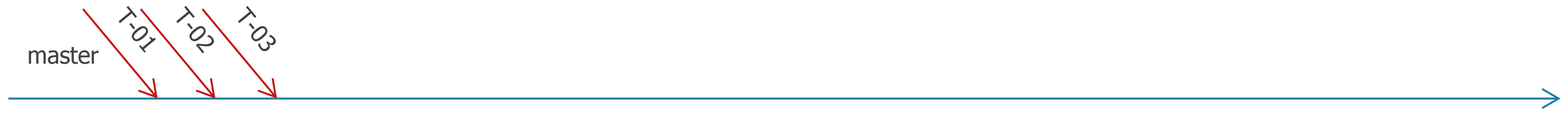
Branches – how to?

- Define a baseline branch for your project (i.e. master, trunk, dev),

Next level - branches

Branches – how to?

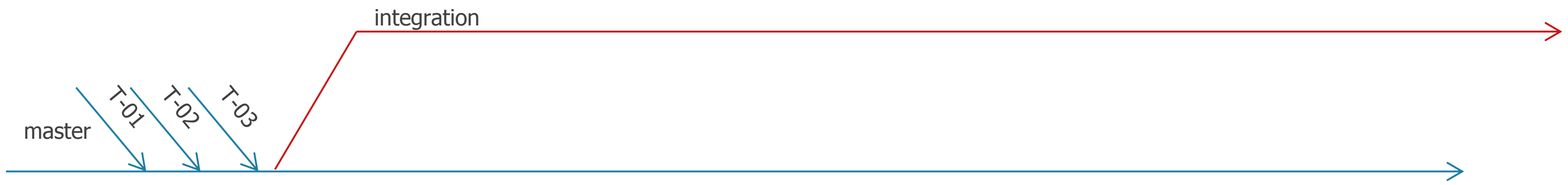
- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master



Next level - branches

Branches – how to?

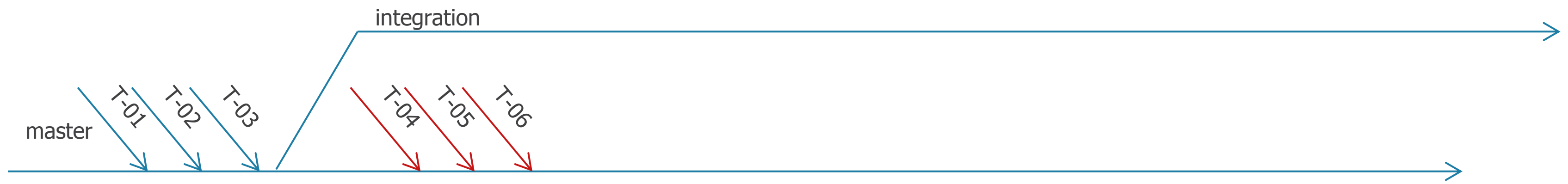
- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master,
- At the end of first development cycle – create an integration branch; it is a good moment to establish cycle deployments;



Next level - branches

Branches – how to?

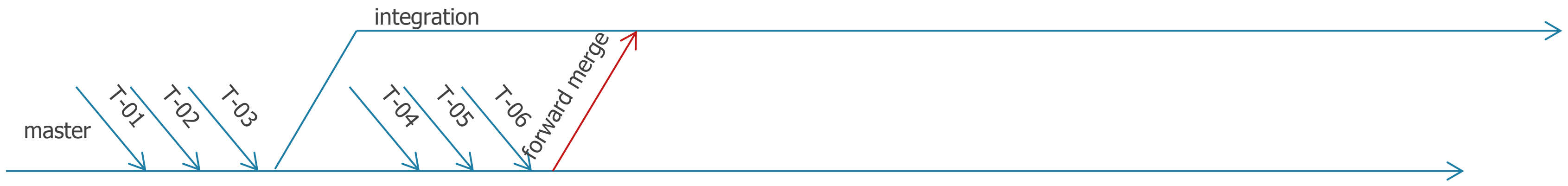
- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master,
- At the end of first development cycle – create an integration branch; it is a good moment to establish cycle deployments;
- Keep working on the dev branch



Next level - branches

Branches – how to?

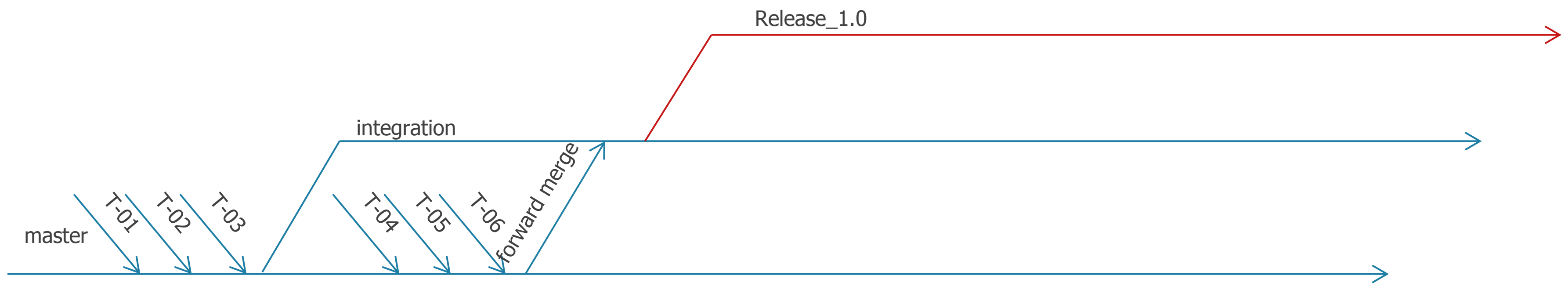
- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master,
- At the end of first development cycle – create an integration branch; it is a good moment to establish cycle deployments;
- Keep working on the dev branch,
- At the end of second development cycle (or any subsequent) – forward changes to integration branch



Next level - branches

Branches – how to?

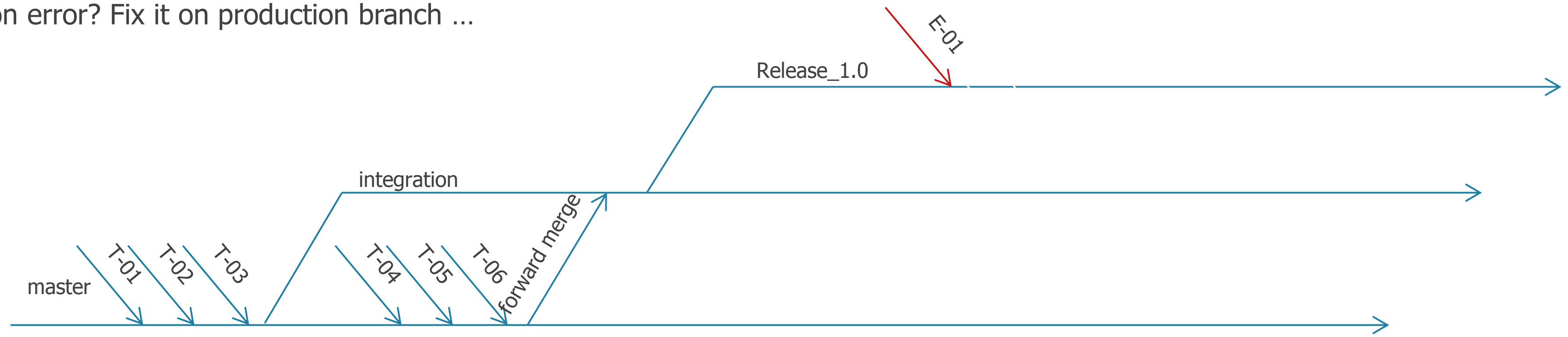
- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master,
- At the end of first development cycle – create an integration branch; it is a good moment to establish cycle deployments;
- Keep working on the dev branch,
- At the end of second development cycle (or any subsequent) – forward changes to integration branch,
- Ready to release? create a release branch; setup a product's SIT and UAT environments;



Next level - branches

Branches – how to?

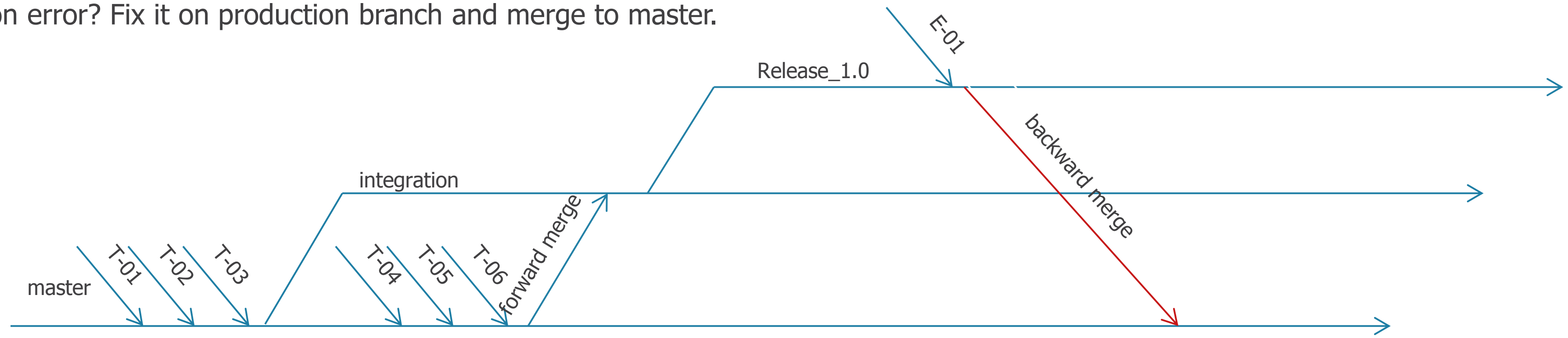
- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master,
- At the end of first development cycle – create an integration branch; it is a good moment to establish cycle deployments;
- Keep working on the dev branch,
- At the end of second development cycle (or any subsequent) – forward changes to integration branch,
- Ready to release? create a release branch; setup a product’s SIT and UAT environments;
- Production error? Fix it on production branch …



Next level - branches

Branches – how to?

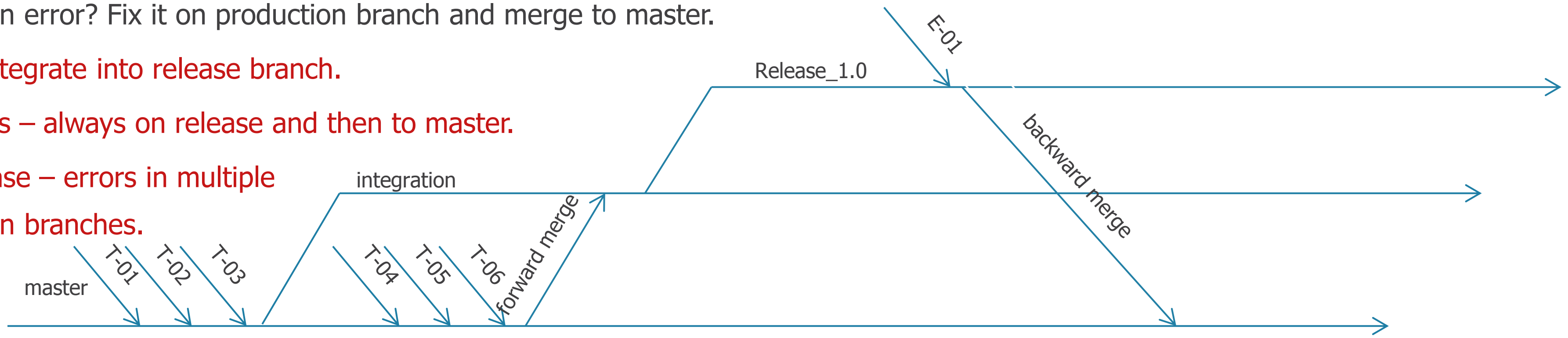
- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master,
- At the end of first development cycle – create an integration branch; it is a good moment to establish cycle deployments;
- Keep working on the dev branch,
- At the end of second development cycle (or any subsequent) – forward changes to integration branch,
- Ready to release? create a release branch; setup a product's SIT and UAT environments;
- Production error? Fix it on production branch and merge to master.



Next level - branches

Branches – how to?

- Define a baseline branch for your project (i.e. master, trunk, dev),
- Start submiting changes to the master,
- At the end of first development cycle – create an integration branch; it is a good moment to establish cycle deployments;
- Keep working on the dev branch,
- At the end of second development cycle (or any subsequent) – forward changes to integration branch,
- Ready to release? create a release branch; setup a product's SIT and UAT environments;
- Production error? Fix it on production branch and merge to master.
- NEVER integrate into release branch.
- Error fixes – always on release and then to master.
- Corner case – errors in multiple production branches.



Exercise

Branches in git hub

- Create a repository, add a single text file, add, commit and push it.
- Create an integration branch, switch to a new branch and submit it

  (`git branch integration, git checkout integration, git push --set-upstream origin integration`)
- Switch back to master branch, make some changes to the file

  (`git checkout master`, ….)
- Merge changes to the integration branch

  (`git checkout integration, git merge master, git push`)
- Branch from integration to release branch

  (`git branch 1.0_rel, git checkout 1.0_rel, git push --set-upstream origin 1.0_rel`)
- Make a change on the master branch (modify first line of the text and submit it)
- Make a „hot fix" on release branch and submit it to release branch (modify first line of text and submit it)
- Merge changes from release branch to master

  (`git checkout master, git merge 1.0_rel`)
- Resolve conflict and submit
- Congratulations – your first release cycle completed!

Multithreading - synchronization

Q & A