Version Control

Basics

Nymgo team

Objectives

Understand basic elements and management of the version control, without entering in details of a specific Version Control System.

Agenda (1h)

1. Basic vocabulary
2. Parallel Worlds
3. Branching Patterns
4. Anti-Patterns

Main concepts

Basic Vocabulary

Environment

- Repository

  Where files' current and historical data are stored.

- Server

  A machine serving the repository.

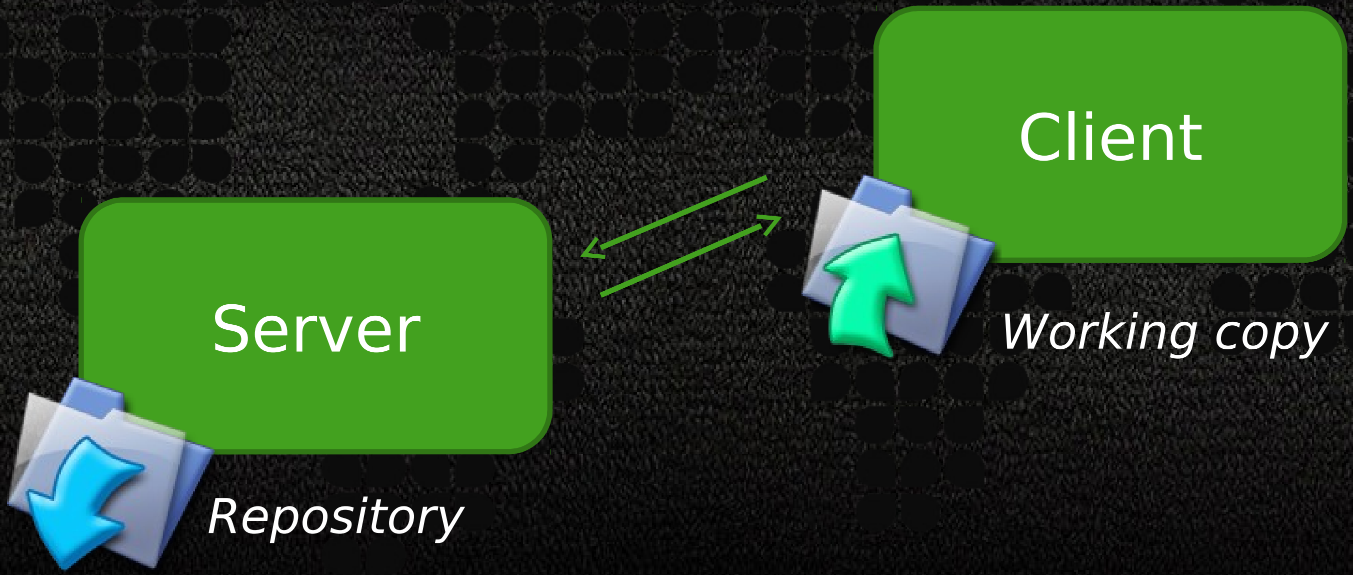
- Client

  The client machine connecting to the server.

- Working copy

  Local copy where the developer changes the code.

Environment



Basic operations

- **Add**

  Mark a file or folder to be versioned

- **Change**

  Create any changes in the local copy

- **Commit**

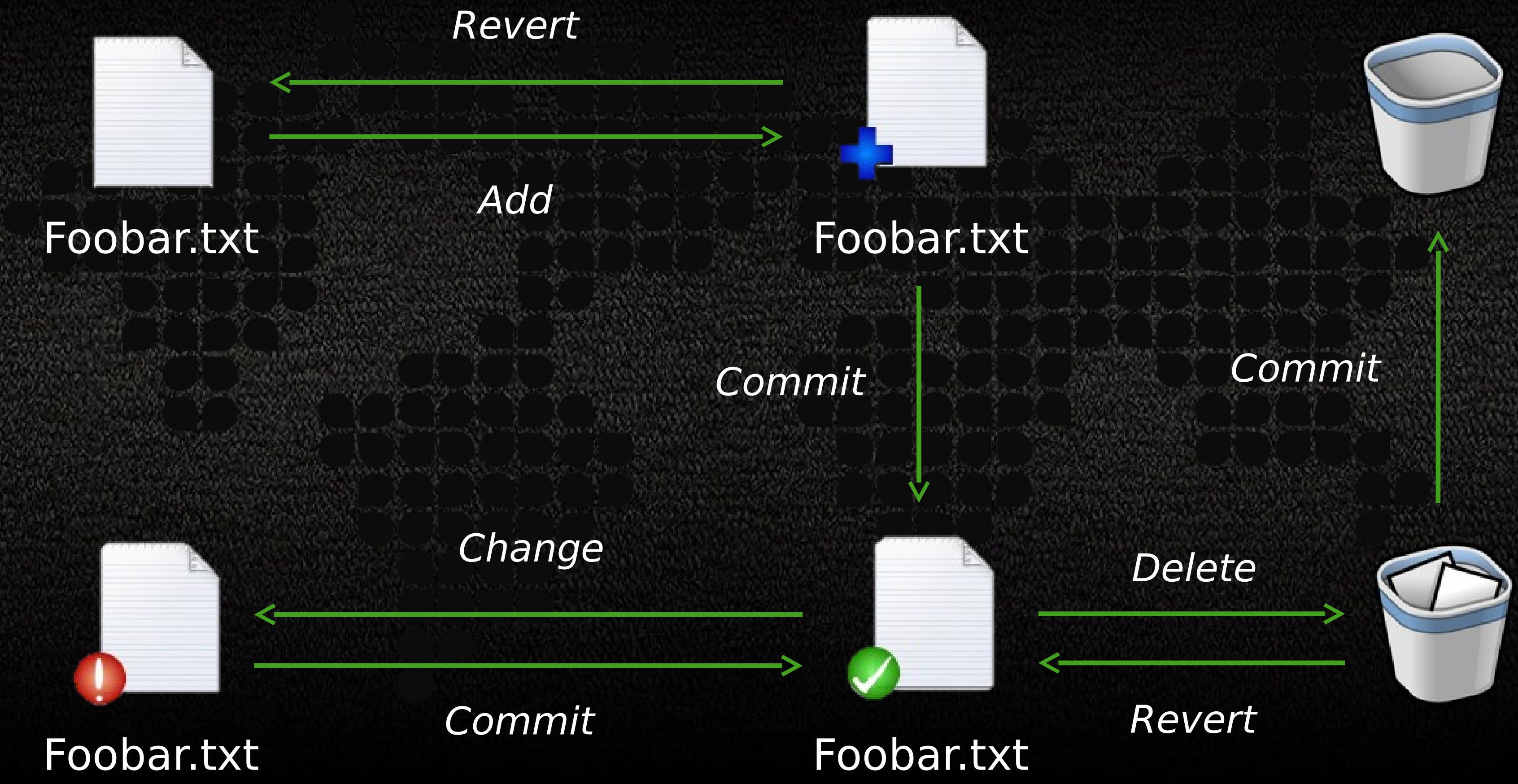
  Send changes to the repository

- **Revert**

  Discard local changes and go back to the same last known revision from the repository

Version Control

Basic operations



Basic artifacts

- Revision

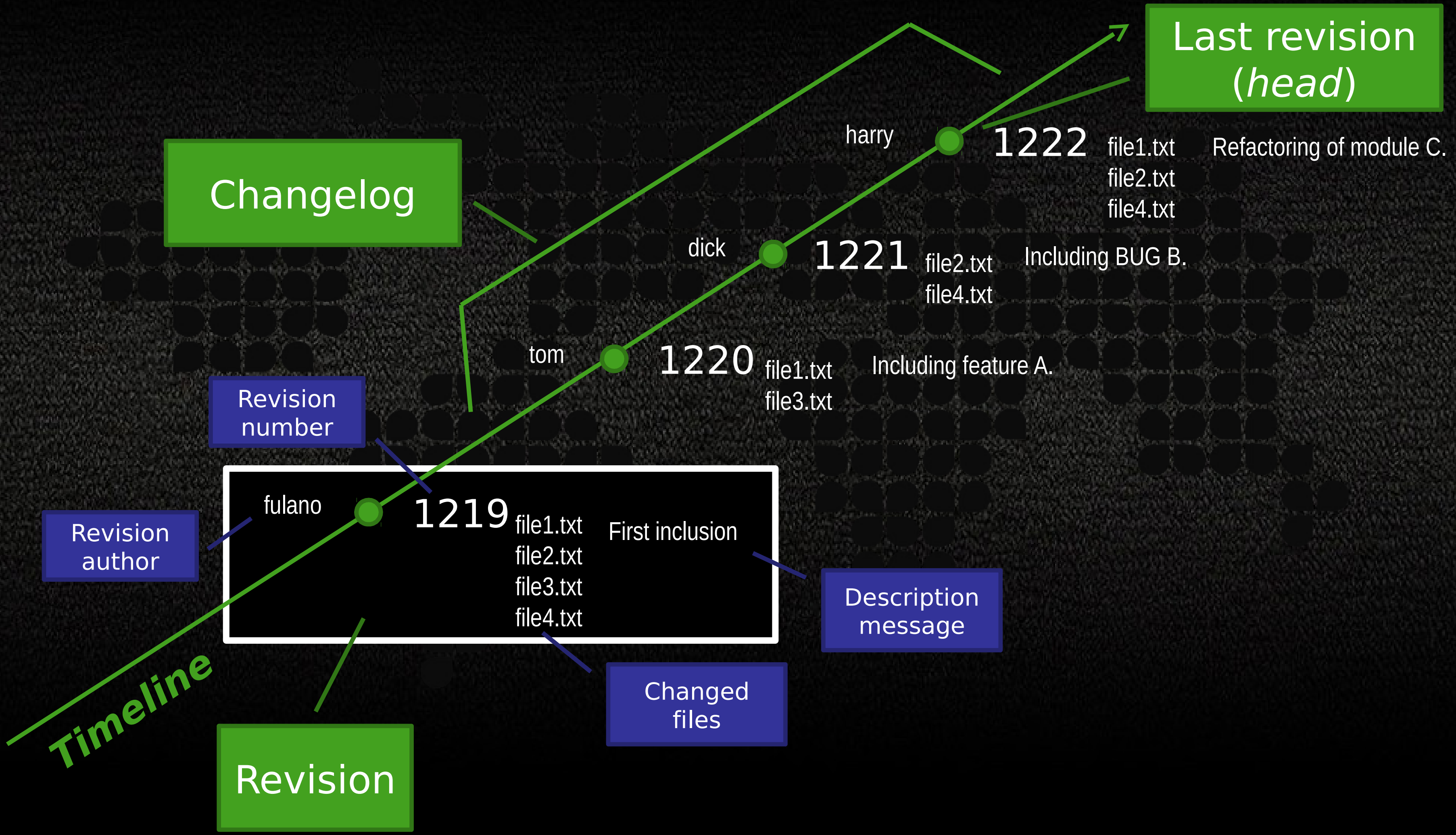
  Set of changes, state of the code in a point of time

- Changelog

  Sequential view of the changes done to the code, stored in the repository

Version Control

Artefatos Básicos

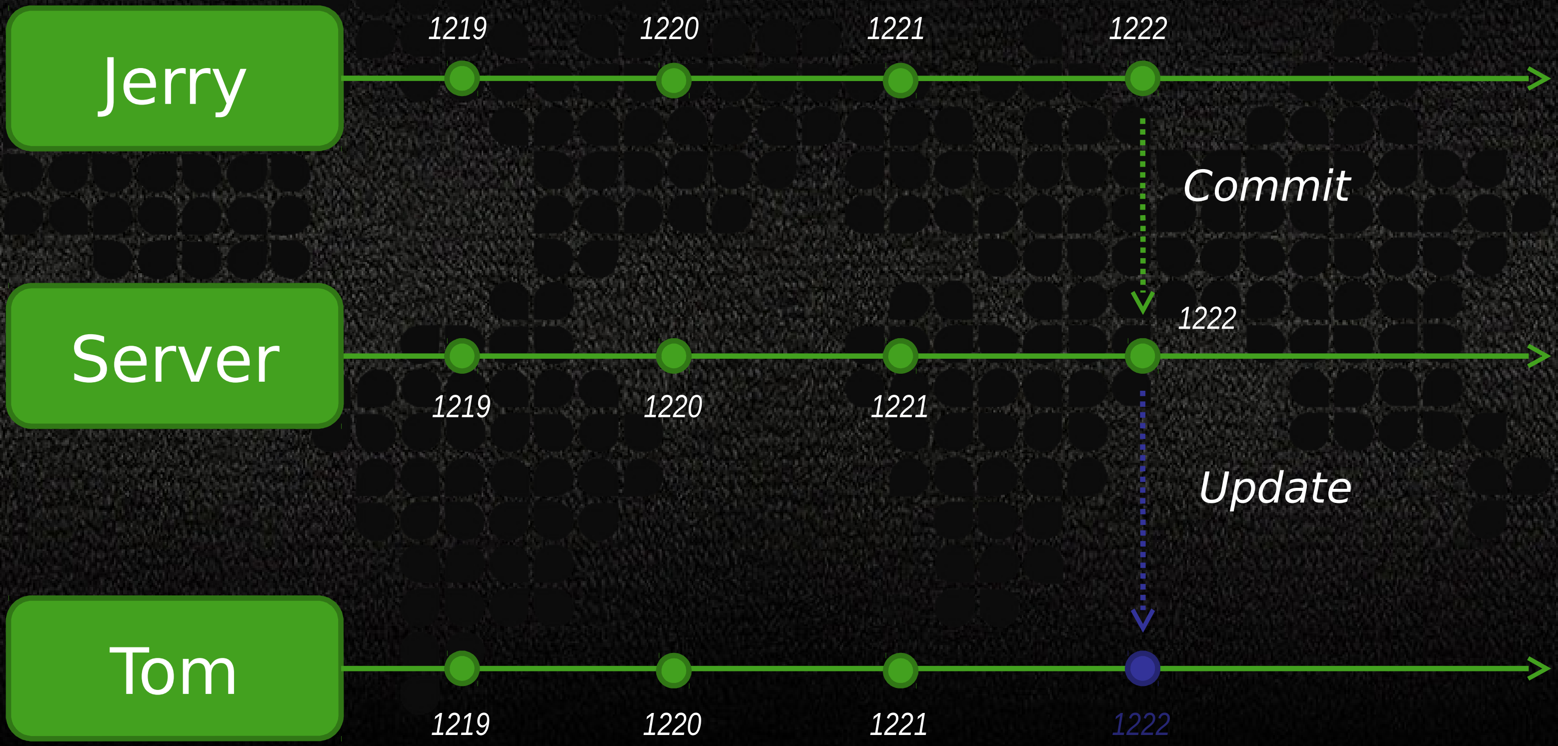


Synchronization

- Update

  Synchronize changes from the repository to the local copy

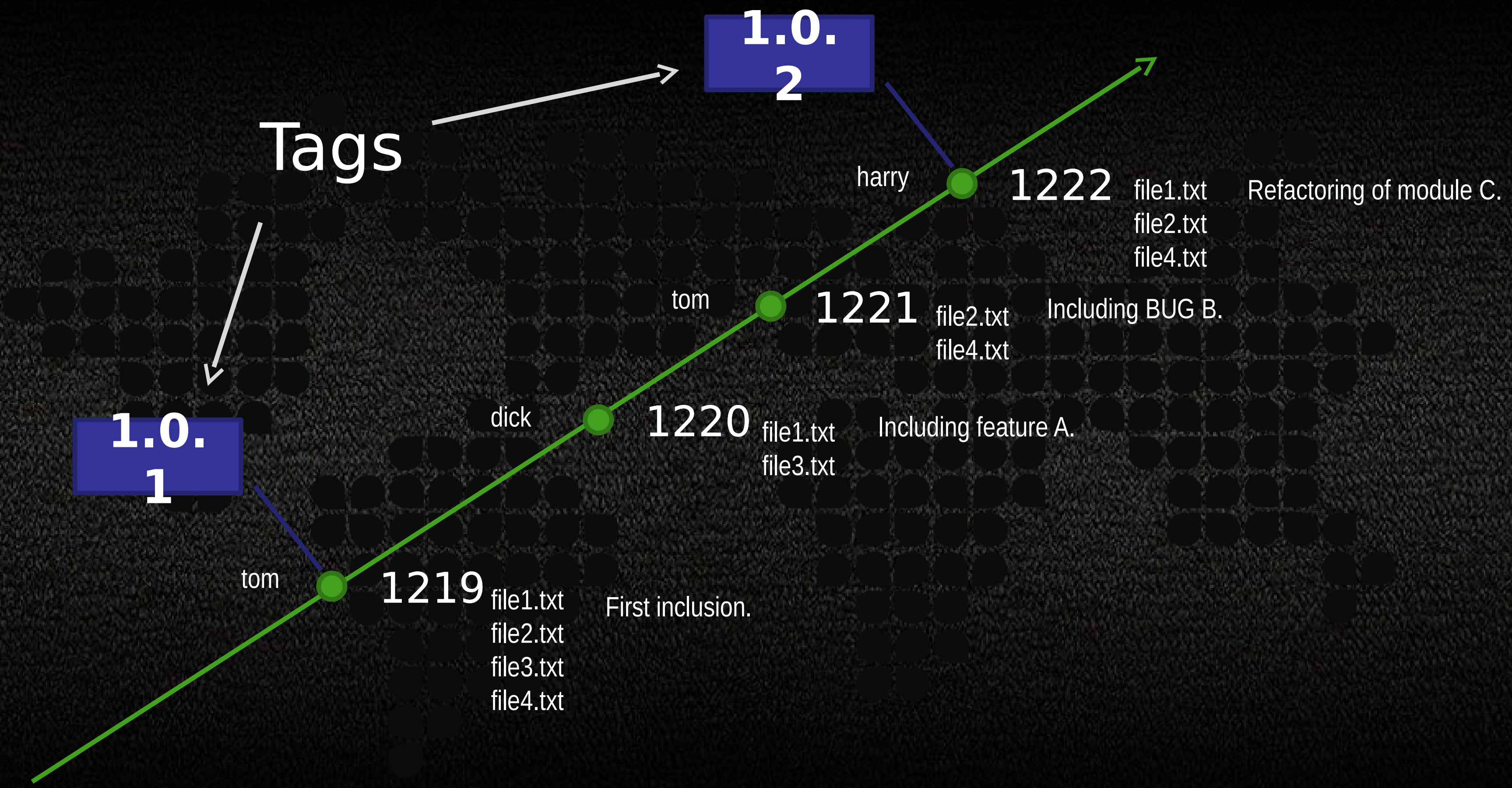
Synchronization



Labels

- Tags
  - Comprehensive alias for existing revisions, marking an important snapshot in time

Labels


1.0.2
Tags
1.0.1
harry
1222
file1.txt
file2.txt
file4.txt
Refactoring of module C.
tom
1221
file2.txt
file4.txt
Including BUG B.
dick
1220
file1.txt
file3.txt
Including feature A.
tom
1219
file1.txt
file2.txt
file3.txt
file4.txt
First inclusion.

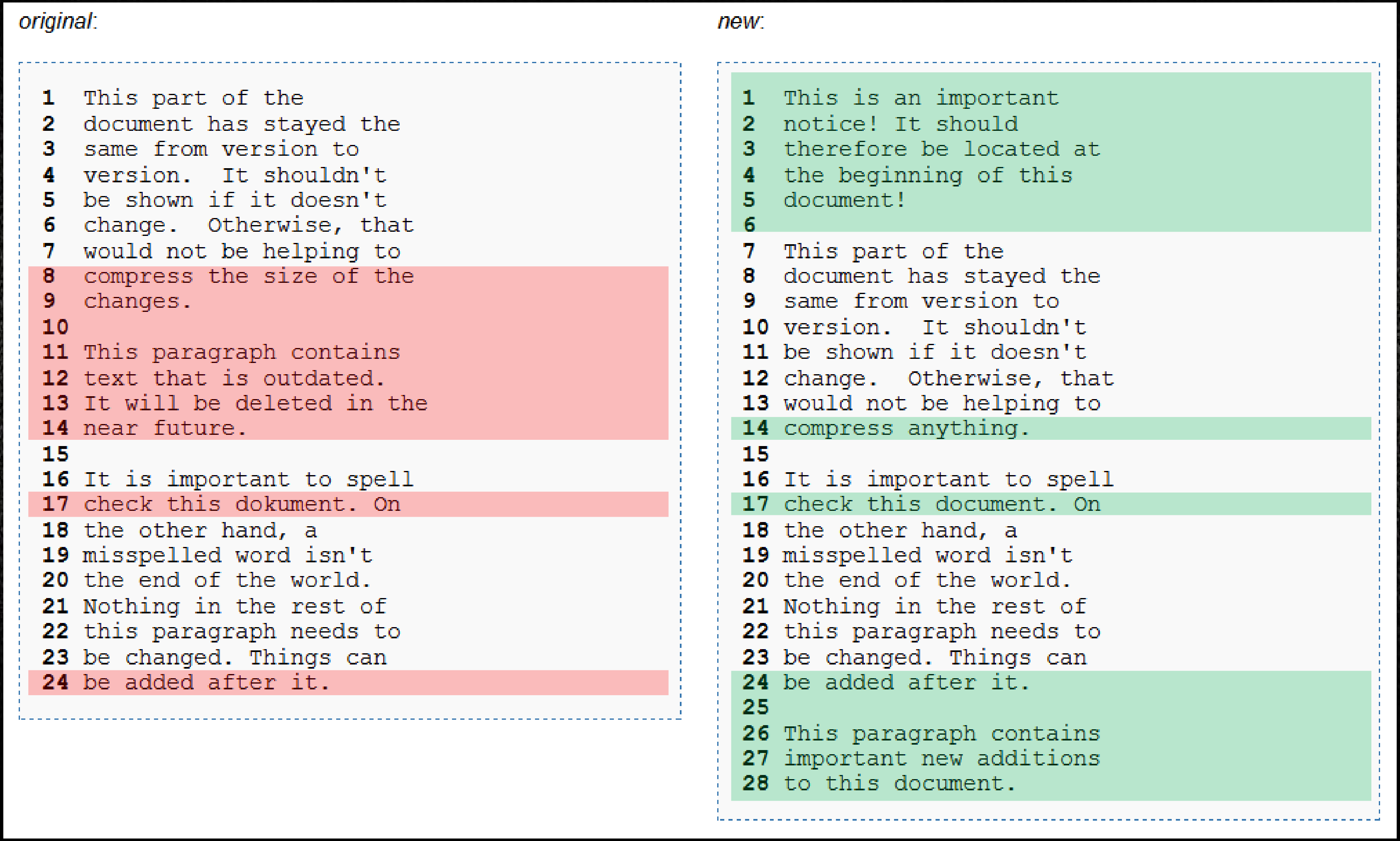
Changes

- Diffs and Patches

The comparison between two text files is done by an application *diff-like* that feeds an application *patch-like* capable of redo the changes from the originating state to the destination state.

This operation of computing the differences is normally referred as *diff*, while the “file containing the differences”, that could be exchanged in e-mails and other eletronic media, is denominated *patch*.

Changes – Example



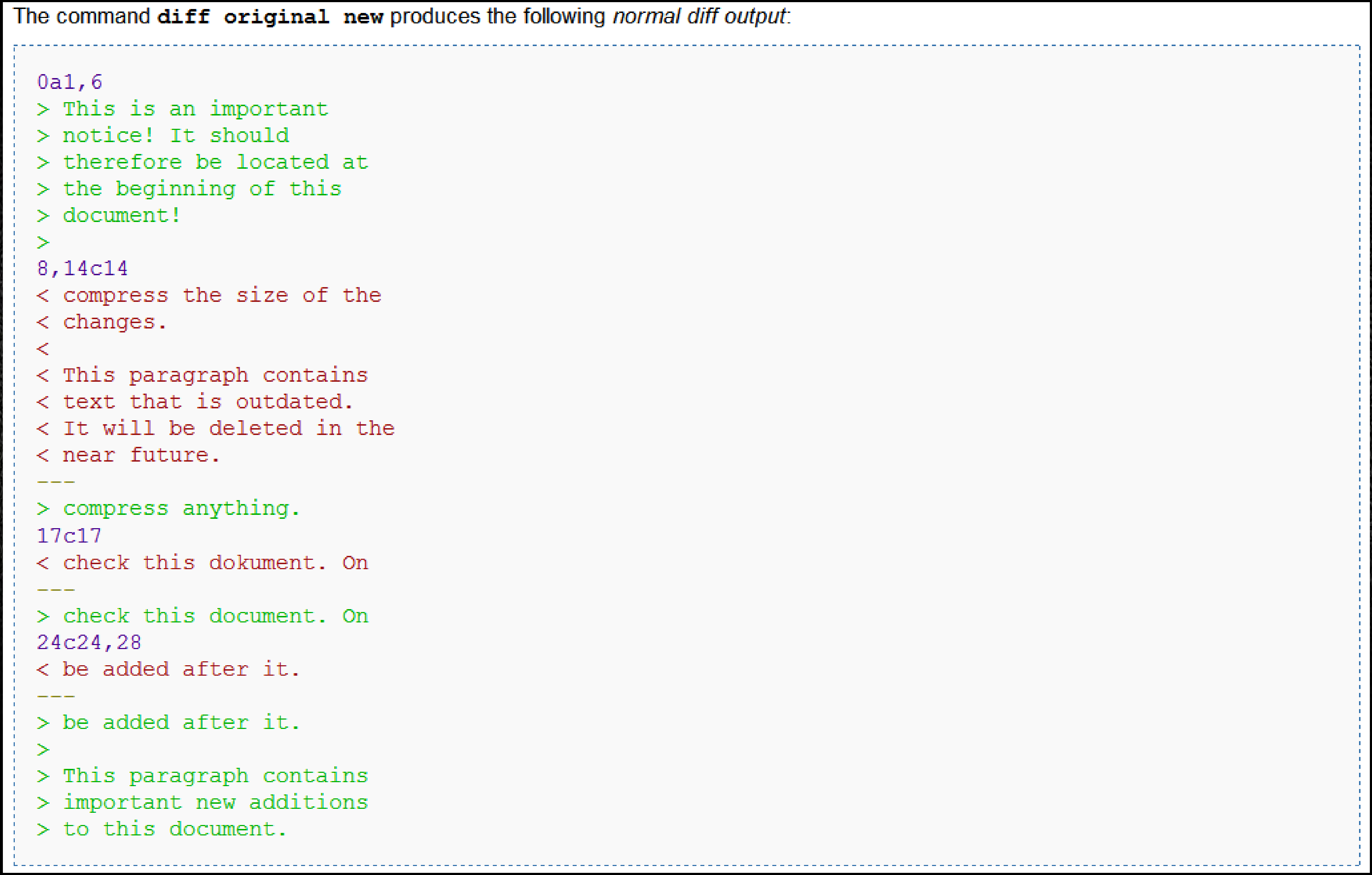
original:

```
1  This part of the
2  document has stayed the
3  same from version to
4  version.  It shouldn't
5  be shown if it doesn't
6  change.  Otherwise, that
7  would not be helping to
8  compress the size of the
9  changes.
10
11 This paragraph contains
12 text that is outdated.
13 It will be deleted in the
14 near future.
15
16 It is important to spell
17 check this dokument. On
18 the other hand, a
19 misspelled word isn't
20 the end of the world.
21 Nothing in the rest of
22 this paragraph needs to
23 be changed. Things can
24 be added after it.
```

new:

```
1  This is an important
2  notice! It should
3  therefore be located at
4  the beginning of this
5  document!
6
7  This part of the
8  document has stayed the
9  same from version to
10 version.  It shouldn't
11 be shown if it doesn't
12 change.  Otherwise, that
13 would not be helping to
14 compress anything.
15
16 It is important to spell
17 check this document. On
18 the other hand, a
19 misspelled word isn't
20 the end of the world.
21 Nothing in the rest of
22 this paragraph needs to
23 be changed. Things can
24 be added after it.
25
26 This paragraph contains
27 important new additions
28 to this document.
```

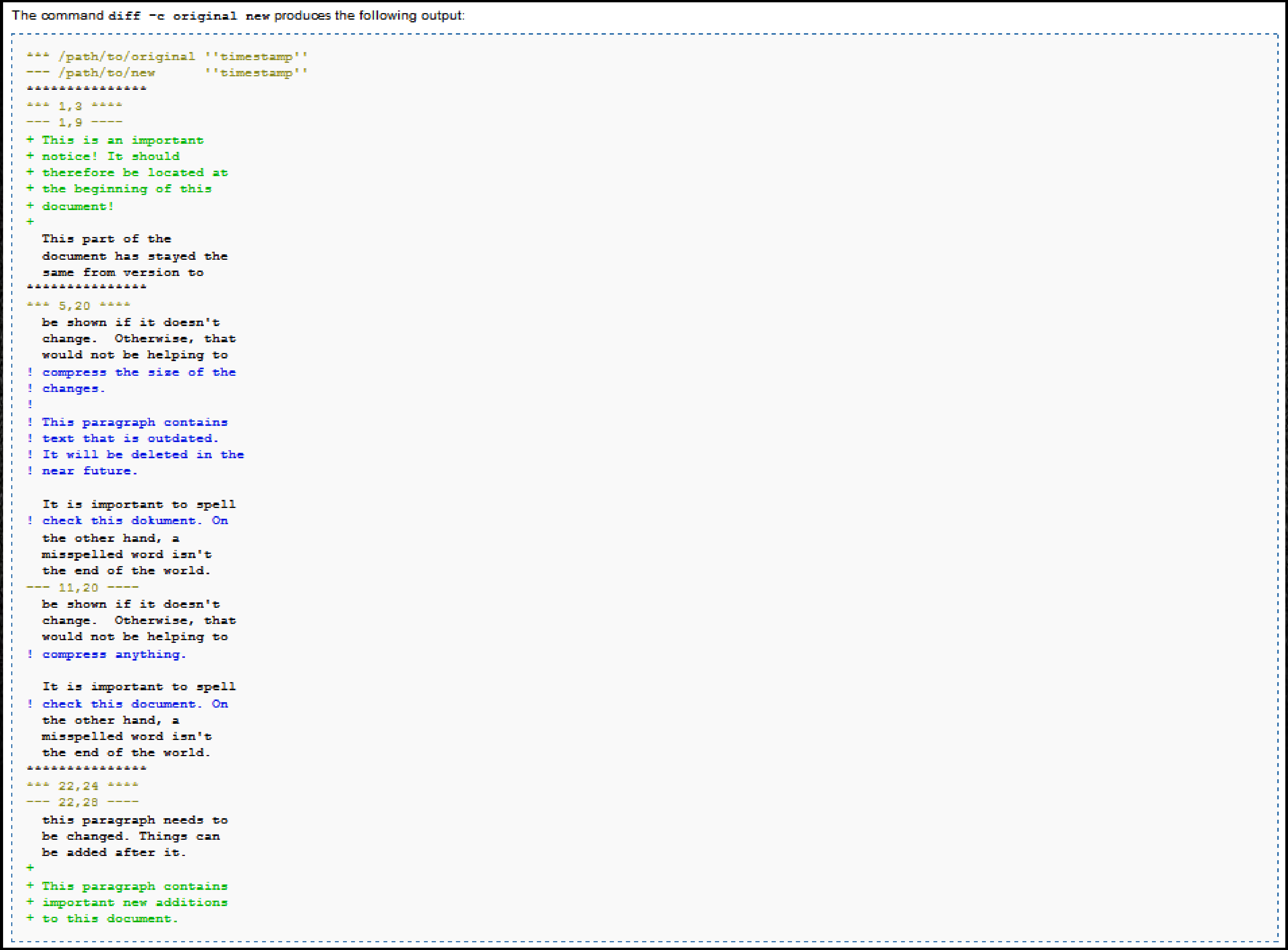
Diff file or patch



The command **`diff original new`** produces the following *normal diff output*:

```
0a1,6
> This is an important
> notice! It should
> therefore be located at
> the beginning of this
> document!
>
8,14c14
< compress the size of the
< changes.
<
< This paragraph contains
< text that is outdated.
< It will be deleted in the
< near future.
---
> compress anything.
17c17
< check this dokument. On
---
> check this document. On
24c24,28
< be added after it.
---
> be added after it.
>
> This paragraph contains
> important new additions
> to this document.
```

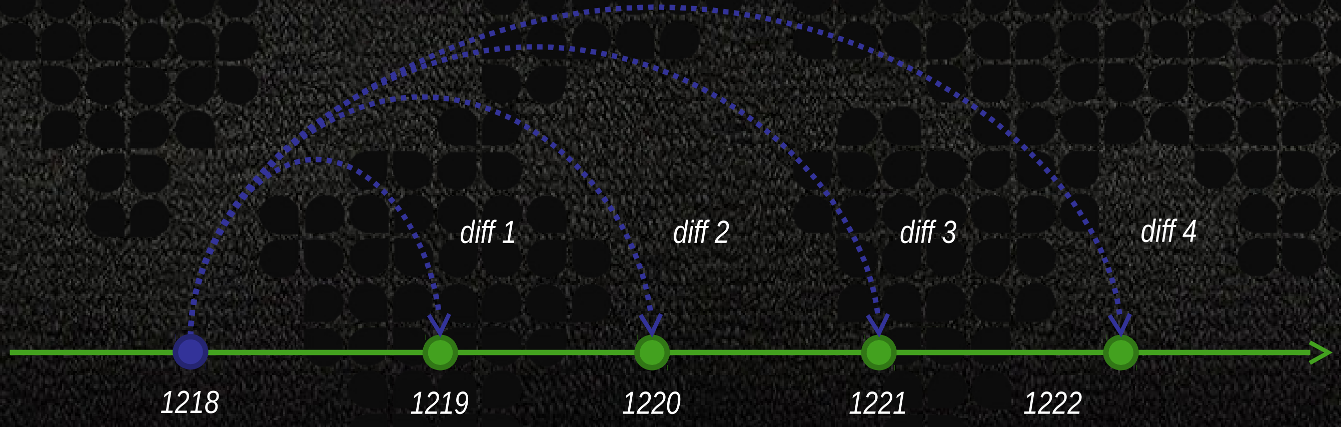
Diff file or patch



The command `diff -c original new` produces the following output:

```
*** /path/to/original ''timestamp''
--- /path/to/new      ''timestamp''
***************
*** 1,3 ****
--- 1,9 ----
+ This is an important
+ notice! It should
+ therefore be located at
+ the beginning of this
+ document!
+
  This part of the
  document has stayed the
  same from version to
***************
*** 5,20 ****
  be shown if it doesn't
  change.  Otherwise, that
  would not be helping to
! compress the size of the
! changes.
!
! This paragraph contains
! text that is outdated.
! It will be deleted in the
! near future.

  It is important to spell
! check this dokument. On
  the other hand, a
  misspelled word isn't
  the end of the world.
--- 11,20 ----
  be shown if it doesn't
  change.  Otherwise, that
  would not be helping to
! compress anything.

  It is important to spell
! check this document. On
  the other hand, a
  misspelled word isn't
  the end of the world.
***************
*** 22,24 ****
--- 22,28 ----
  this paragraph needs to
  be changed. Things can
  be added after it.
+
+ This paragraph contains
+ important new additions
+ to this document.
```

Changes

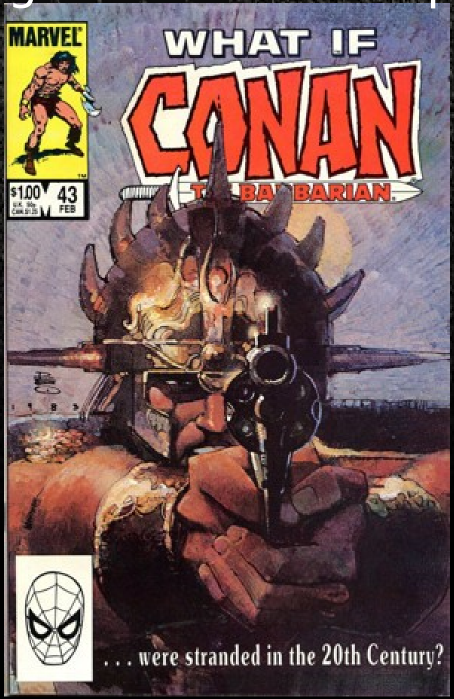


Concepts of branching, merging and conflicts

Parallel Worlds

Parallel Worlds

Perhaps the most accessible way to think of branches is as parallel universes. They're places where, for whatever reason, history didn't go quite the same way as it did in your universe. From that point forward, that universe can be slightly different – or it can be radically and utterly transformed. Like the Marvel comic book series “What If?”, branching lets you answer some interesting and possibly even dangerous "what if" questions with your software development.



WHAT IF
MARVEL COMICS GROUP
50¢ 1 FEB
WHAT IF SPIDER-MAN JOINED THE Fantastic Four?
ALL NEW COVER TO COVER!
WAIT'LL THEY MEET THE FANTASTIC FIVE!

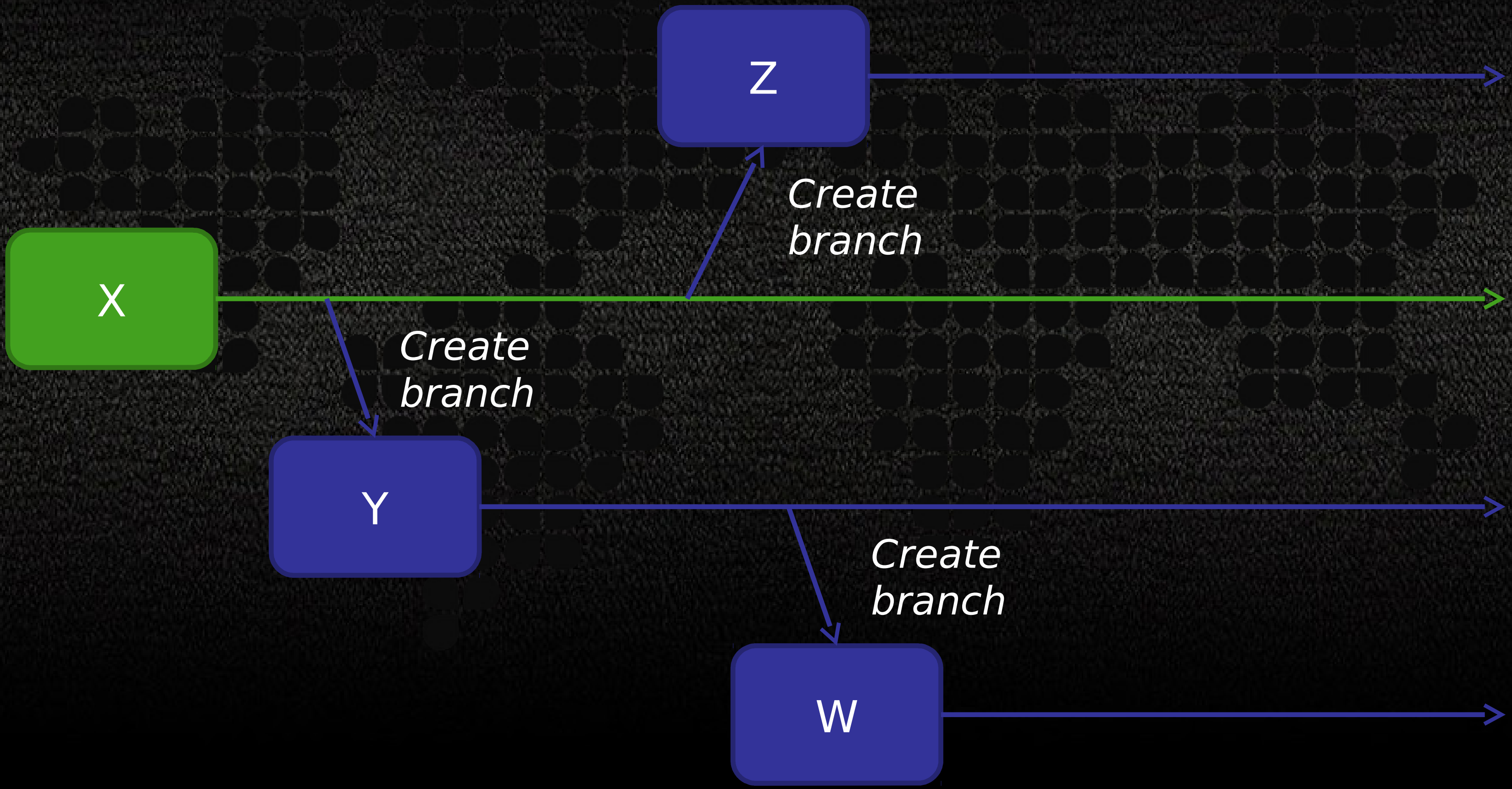
Parallel Worlds

- Parallel universes offer unlimited possibilities.
- They also allow you to stay safely ensconced in the particular universe of your choice, completely isolated from any events in other alternate universes.

Parallel Worlds

- Branch is like a parallel world.
- Changes done in one branch doesn’t have effect over the another.
- Although branching offers the seductive appeal of multiple possibility with very little risk, it also brings along something far less desirable: **increase of complexity**.

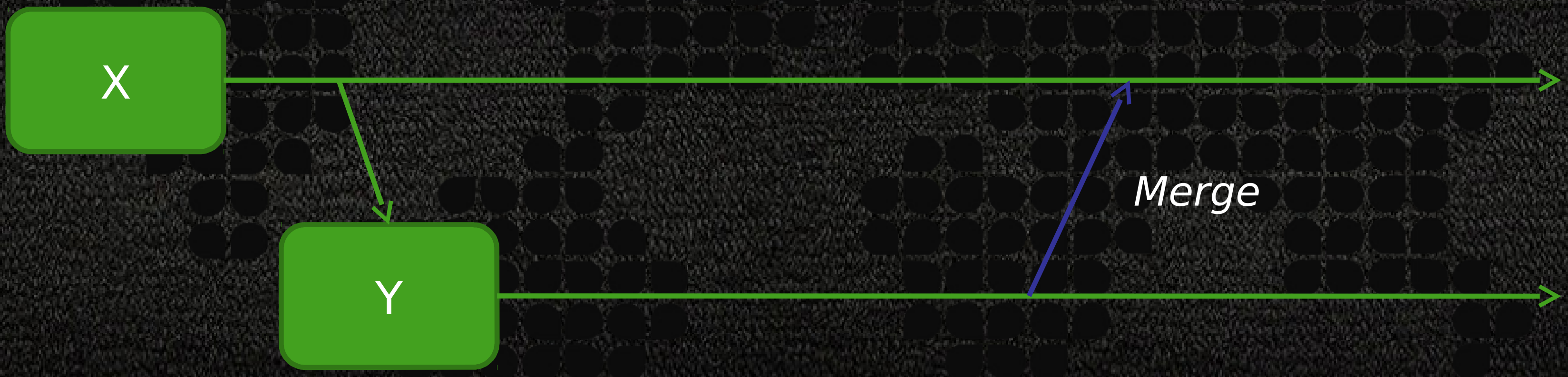
Branches



Merge

- In some point of the development, the code must be merged.

Merge



Conflicts

- When the changes are done in the same source file lines, you can have conflicts.

How to solve conflicts?

How to solve conflicts?

Question:

Is it possible to solve these conflicts automatically?

How to solve conflicts?

Answer: No!!!

Even when don't have changes in the same source line, you can still have semantic problems in the code. So, always double check before to commit.

How to solve conflicts?

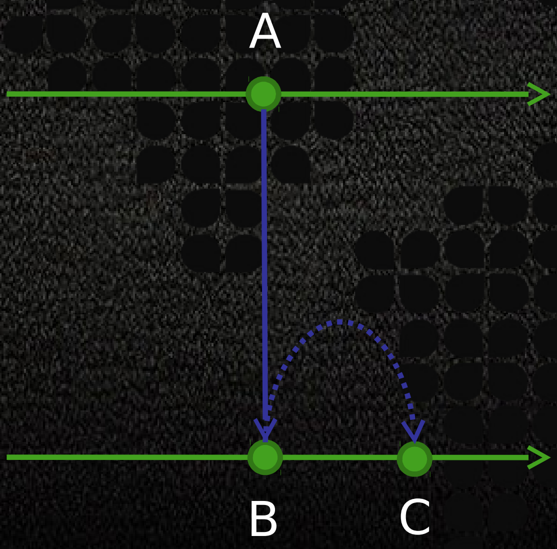
A note before continuing:

It is always better to avoid unnecessary conflicts.

Communicate your development, always include a description in your committed revisions, ask for help if you don’t understand anything, etc.

How to solve conflicts?

Two way merge

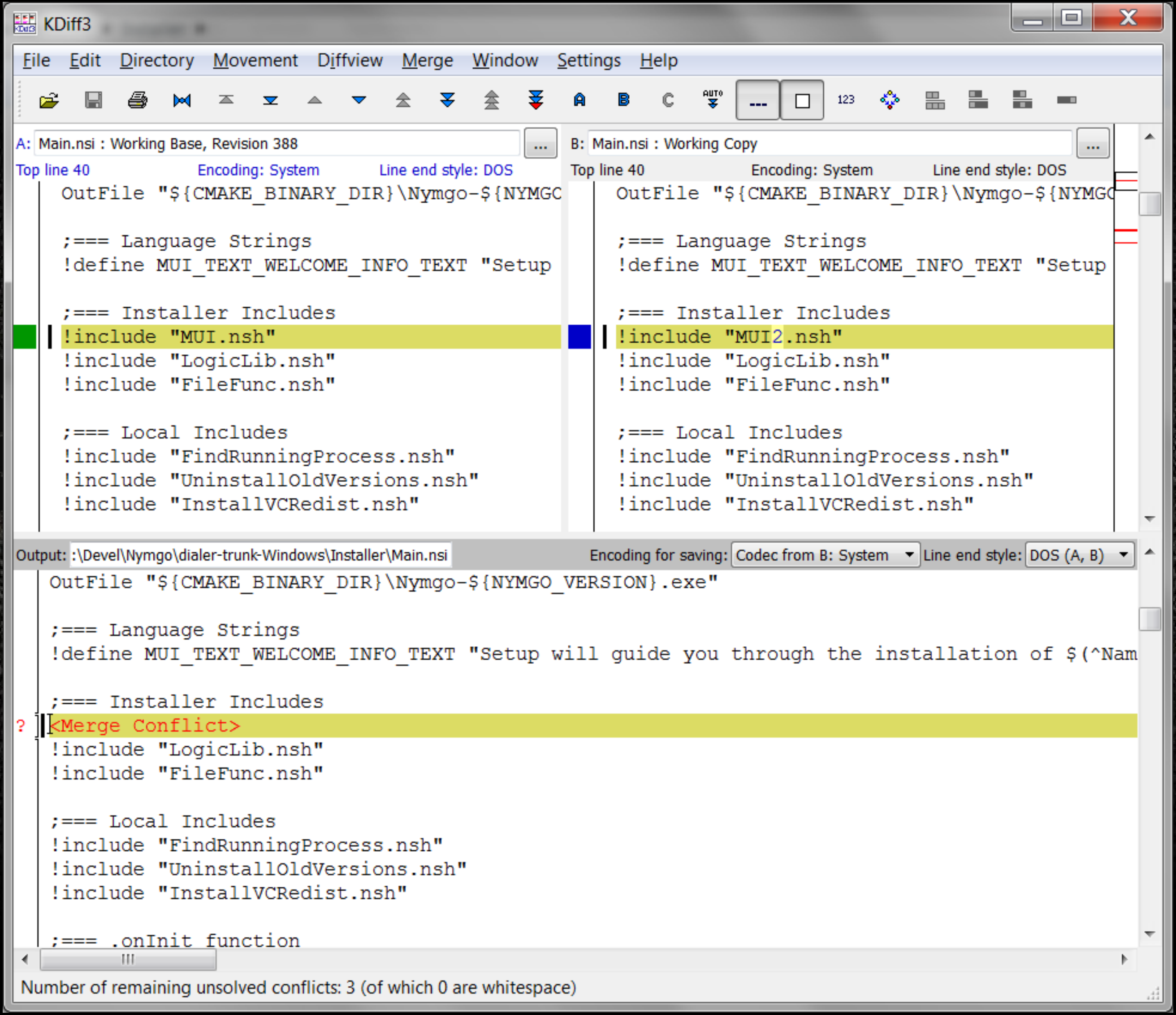


A working base

B working copy

C = A + B output

Tools: KDiff3


KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help

A: Main.nsi : Working Base, Revision 388
Top line 40 Encoding: System Line end style: DOS

```
OutFile "${CMAKE_BINARY_DIR}\Nymgo-${NYMGC

;=== Language Strings
!define MUI_TEXT_WELCOME_INFO_TEXT "Setup

;=== Installer Includes
!include "MUI.nsh"
!include "LogicLib.nsh"
!include "FileFunc.nsh"

;=== Local Includes
!include "FindRunningProcess.nsh"
!include "UninstallOldVersions.nsh"
!include "InstallVCRedist.nsh"
```

B: Main.nsi : Working Copy
Top line 40 Encoding: System Line end style: DOS

```
OutFile "${CMAKE_BINARY_DIR}\Nymgo-${NYMGC

;=== Language Strings
!define MUI_TEXT_WELCOME_INFO_TEXT "Setup

;=== Installer Includes
!include "MUI2.nsh"
!include "LogicLib.nsh"
!include "FileFunc.nsh"

;=== Local Includes
!include "FindRunningProcess.nsh"
!include "UninstallOldVersions.nsh"
!include "InstallVCRedist.nsh"
```

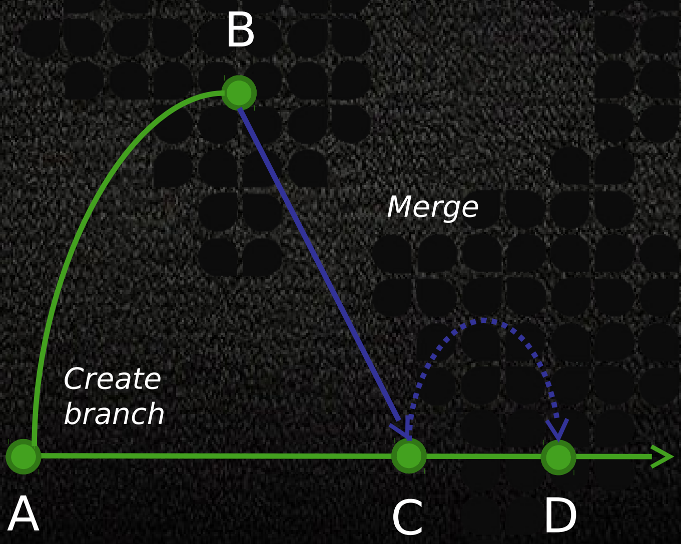
Output: :\Devel\Nymgo\dialer-trunk-Windows\Installer\Main.nsi Encoding for saving: Codec from B: System Line end style: DOS (A, B)

```
OutFile "${CMAKE_BINARY_DIR}\Nymgo-${NYMGO_VERSION}.exe"

;=== Language Strings
!define MUI_TEXT_WELCOME_INFO_TEXT "Setup will guide you through the installation of $(^Nam

;=== Installer Includes
<Merge Conflict>
!include "LogicLib.nsh"
!include "FileFunc.nsh"

;=== Local Includes
!include "FindRunningProcess.nsh"
!include "UninstallOldVersions.nsh"
!include "InstallVCRedist.nsh"

;=== .onInit function
```

Number of remaining unsolved conflicts: 3 (of which 0 are whitespace)

How to solve conflicts?

Three way merge



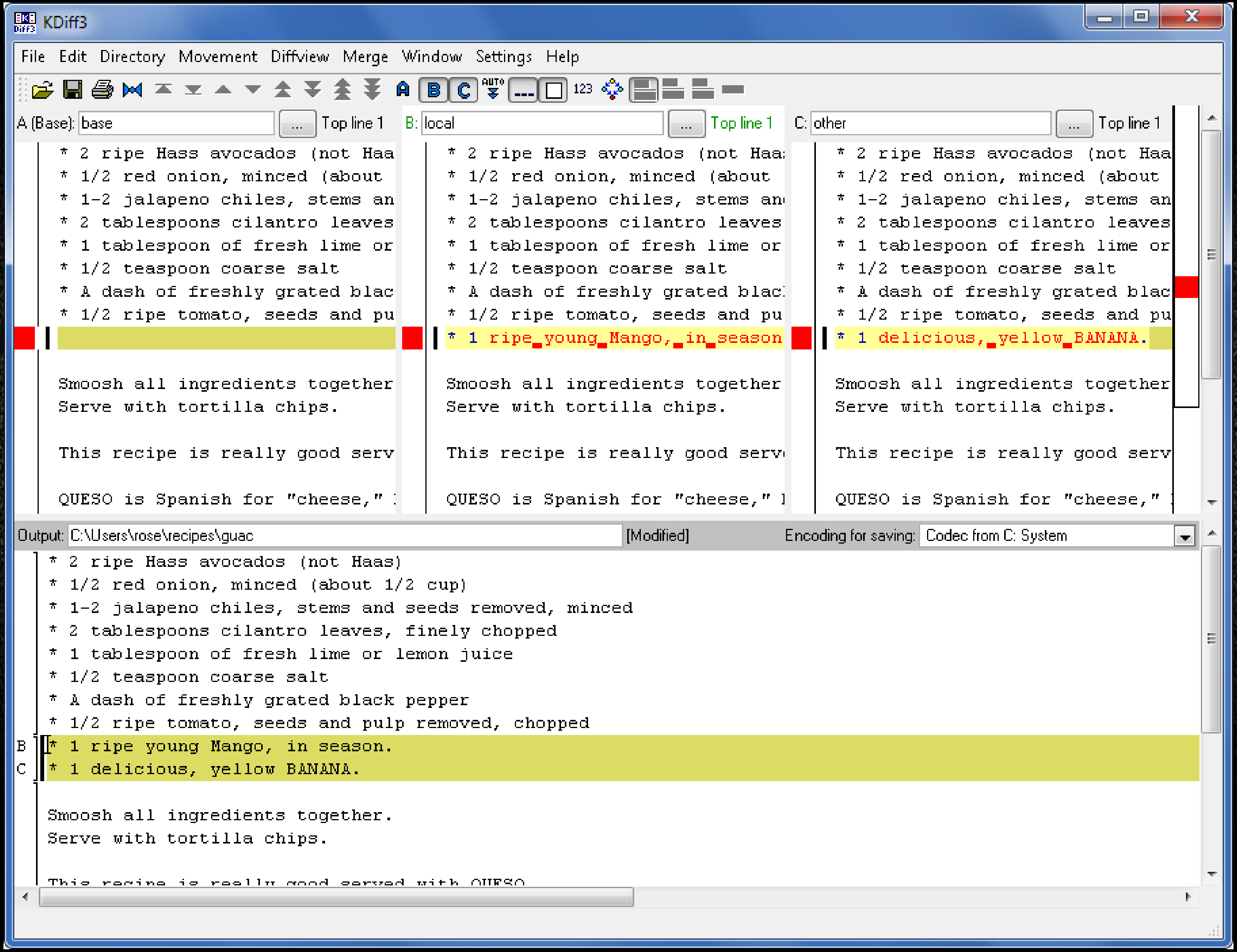
A origin

B branch #1

C branch #2

D = A + B + C

Tools: KDiff3


KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
A (Base): base
Top line 1
B: local
Top line 1
C: other
Top line 1
Output: C:\Users\rose\recipes\guac
[Modified]
Encoding for saving: Codec from C: System

```
A (Base): base
* 2 ripe Hass avocados (not Haa
* 1/2 red onion, minced (about
* 1-2 jalapeno chiles, stems an
* 2 tablespoons cilantro leaves
* 1 tablespoon of fresh lime or
* 1/2 teaspoon coarse salt
* A dash of freshly grated blac
* 1/2 ripe tomato, seeds and pu


Smoosh all ingredients together
Serve with tortilla chips.

This recipe is really good serv

QUESO is Spanish for "cheese," 
```

```
B: local
* 2 ripe Hass avocados (not Haa
* 1/2 red onion, minced (about
* 1-2 jalapeno chiles, stems an
* 2 tablespoons cilantro leaves
* 1 tablespoon of fresh lime or
* 1/2 teaspoon coarse salt
* A dash of freshly grated blac
* 1/2 ripe tomato, seeds and pu
* 1 ripe young Mango, in season

Smoosh all ingredients together
Serve with tortilla chips.

This recipe is really good serv

QUESO is Spanish for "cheese," 
```

```
C: other
* 2 ripe Hass avocados (not Haa
* 1/2 red onion, minced (about
* 1-2 jalapeno chiles, stems an
* 2 tablespoons cilantro leaves
* 1 tablespoon of fresh lime or
* 1/2 teaspoon coarse salt
* A dash of freshly grated blac
* 1/2 ripe tomato, seeds and pu
* 1 delicious, yellow BANANA.

Smoosh all ingredients together
Serve with tortilla chips.

This recipe is really good serv

QUESO is Spanish for "cheese," 
```

```
Output: C:\Users\rose\recipes\guac
  * 2 ripe Hass avocados (not Haas)
  * 1/2 red onion, minced (about 1/2 cup)
  * 1-2 jalapeno chiles, stems and seeds removed, minced
  * 2 tablespoons cilantro leaves, finely chopped
  * 1 tablespoon of fresh lime or lemon juice
  * 1/2 teaspoon coarse salt
  * A dash of freshly grated black pepper
  * 1/2 ripe tomato, seeds and pulp removed, chopped
B * 1 ripe young Mango, in season.
C * 1 delicious, yellow BANANA.

  Smoosh all ingredients together.
  Serve with tortilla chips.

  This recipe is really good served with QUESO
```

Tools: choose one...

Meld

Guiffy SureMerge

Ultracompare

DiffMerge

Notepad++

Apple Filemerge

Araxis

tkmerge

TFS diffmerge

xxdiff

...

Which one should I use?

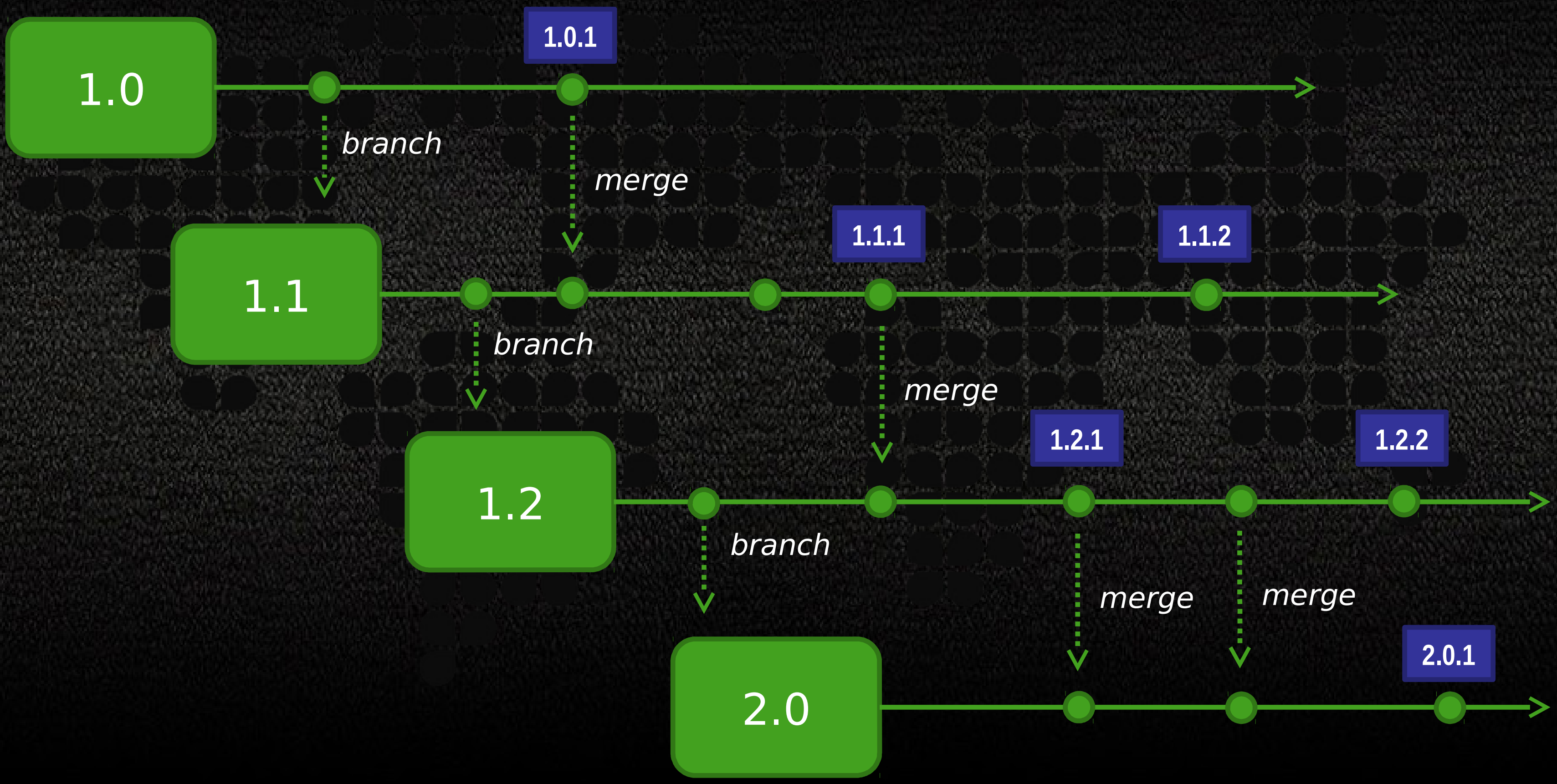
Choose one that best fits your team!

How to manage the branches during the software lifecycle

Branching Patterns

Version Control

Mainline



Mainline

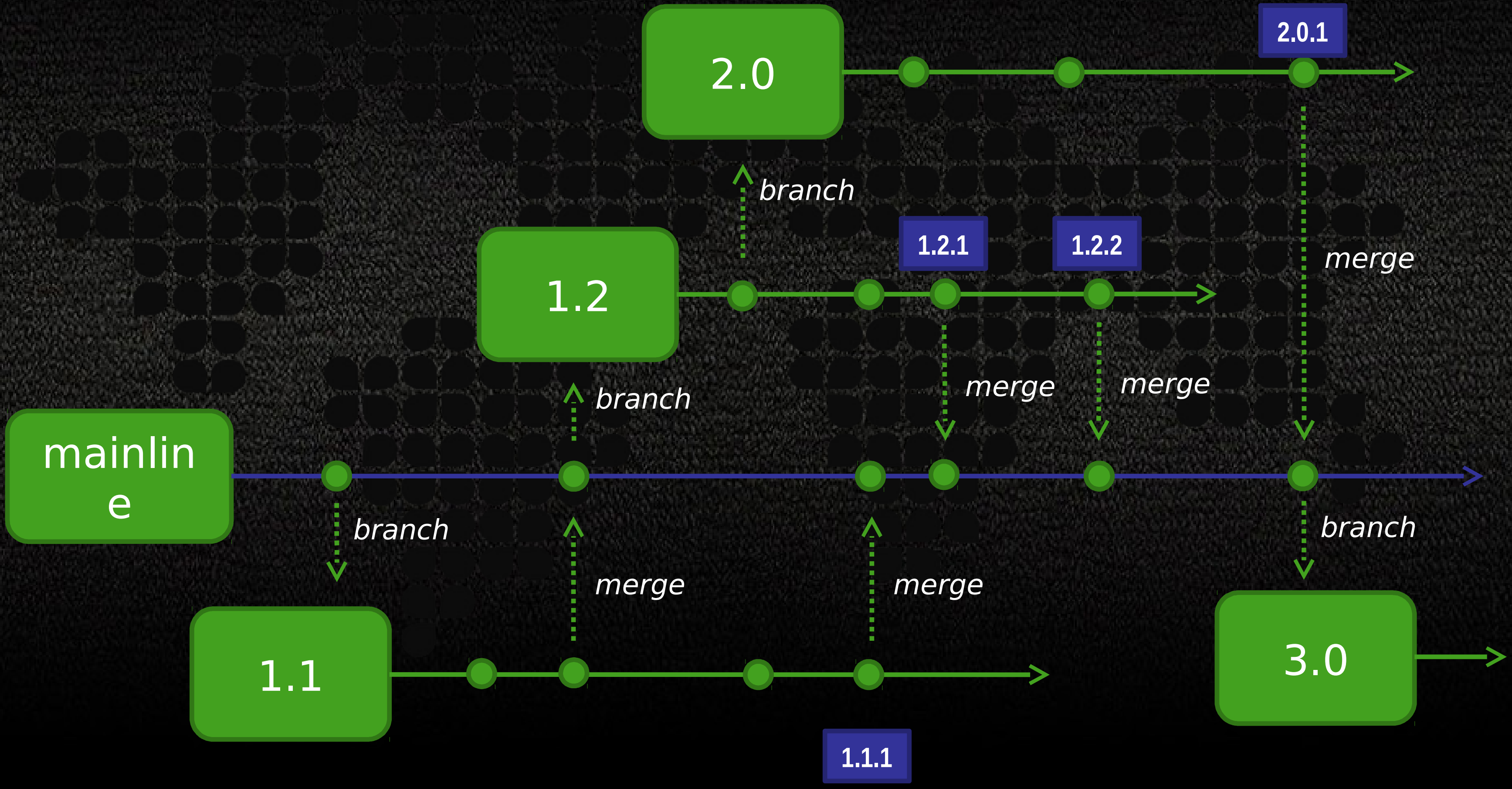
Bring a fix from branch *n* to branch *m* require *m-n* merges (*linear complexity with the number of branches*).

Mainline

Learned lesson:

Perform merges the early and frequently as possible.

Mainline / Variant



Version Control

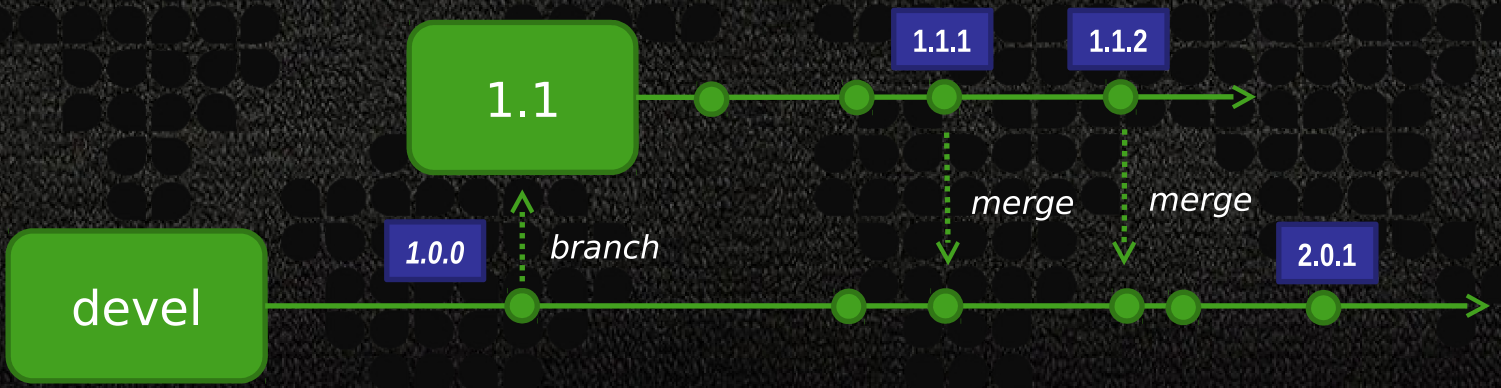
Scenario:

You just developed a stable version of the software and you need to create a new version with new features and still provide small fixes for the last stable version.

Version Control

Parallel Maintenance / Development Lines

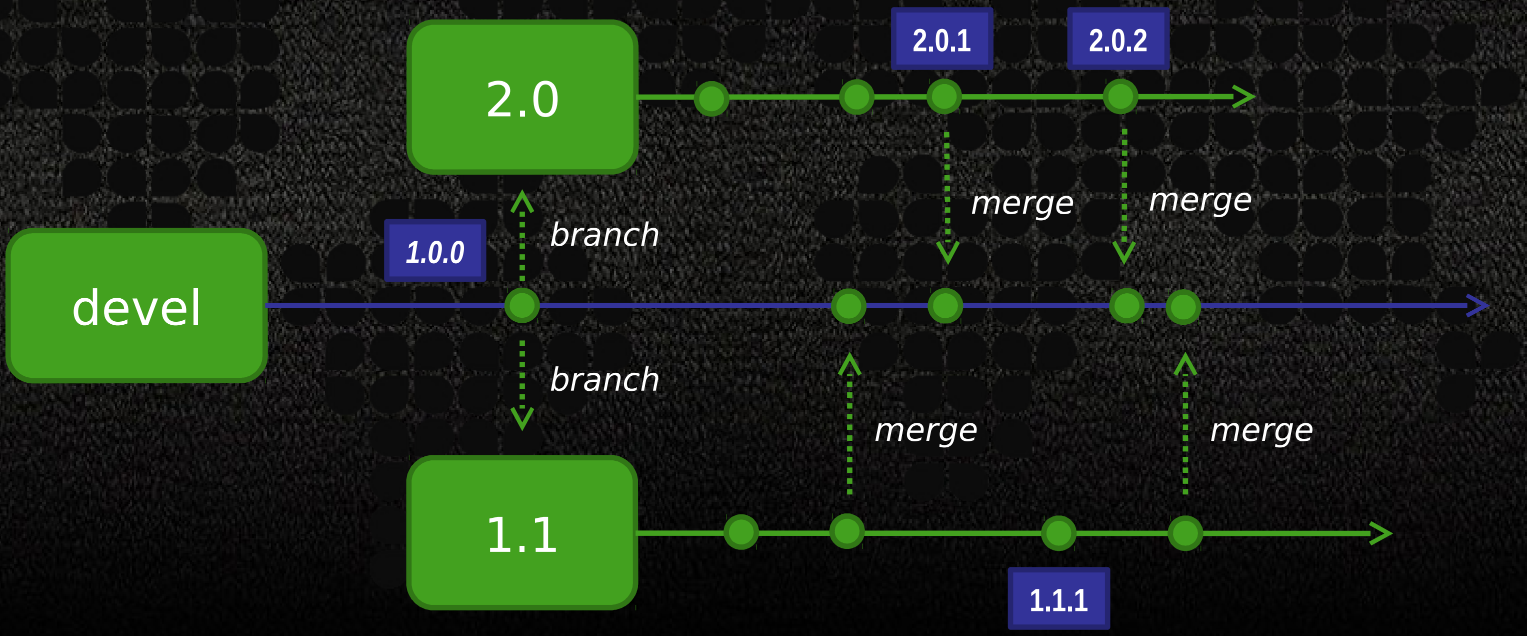
Most used option



Version Control

Parallel Maintenance / Development Lines

Note the internal branch



Version Control

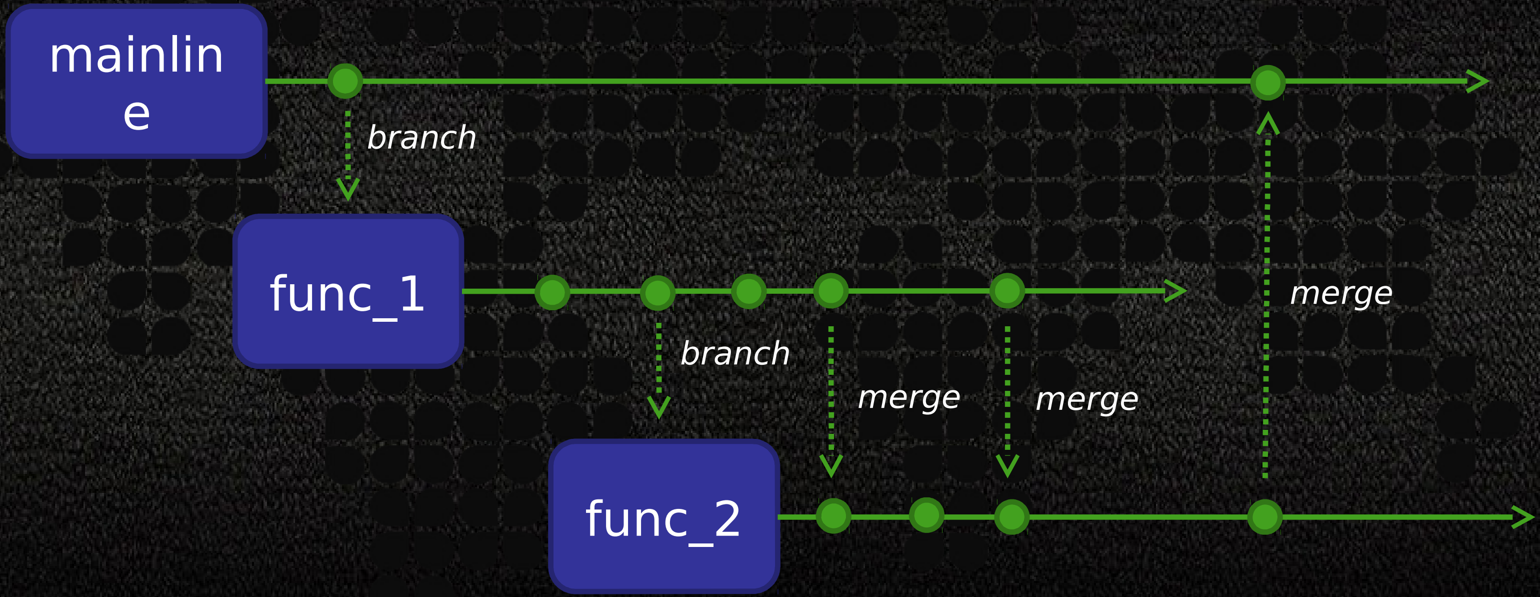
Scenario:

You need to develop two different features in a short period of time.

(laughts)

Version Control

Overlapping Release Lines



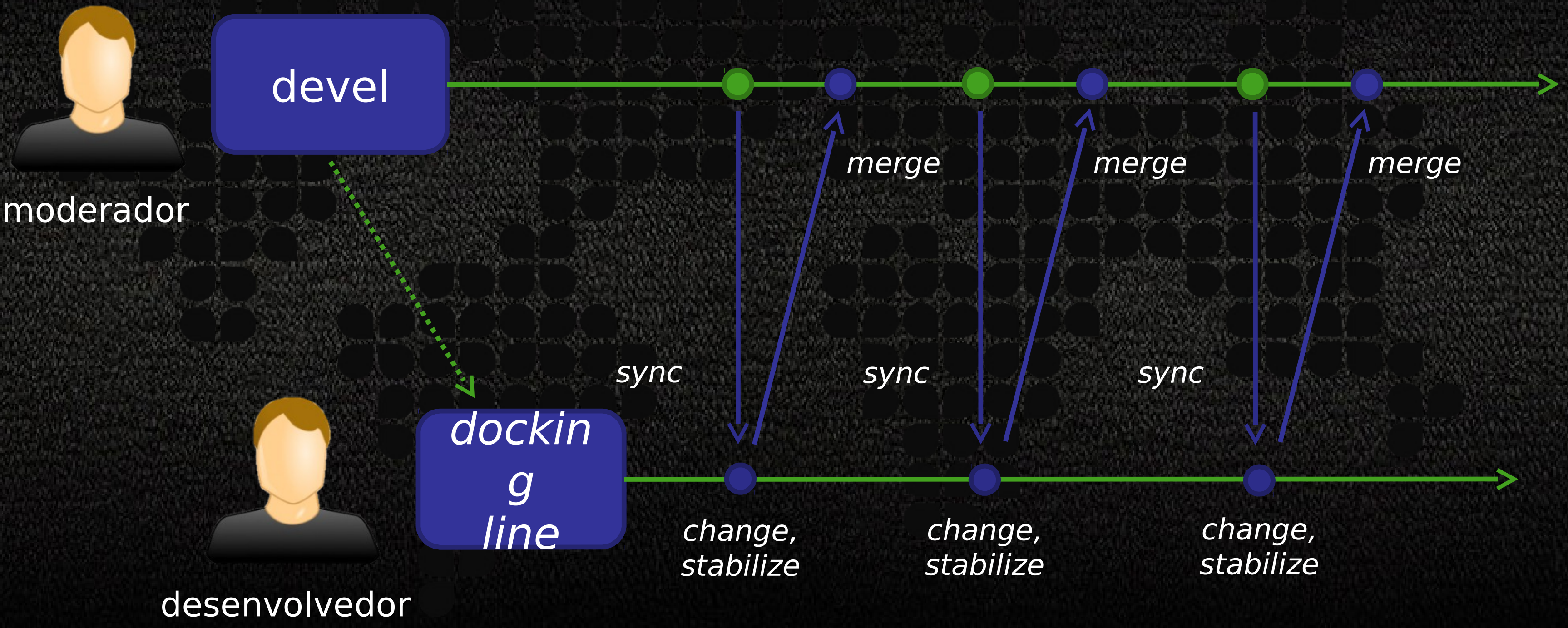
Version Control

Scenario:

You allocated a new developer to work in a too risky code base and you want to moderate his/her changes for a while.

Version Control

Docking Line

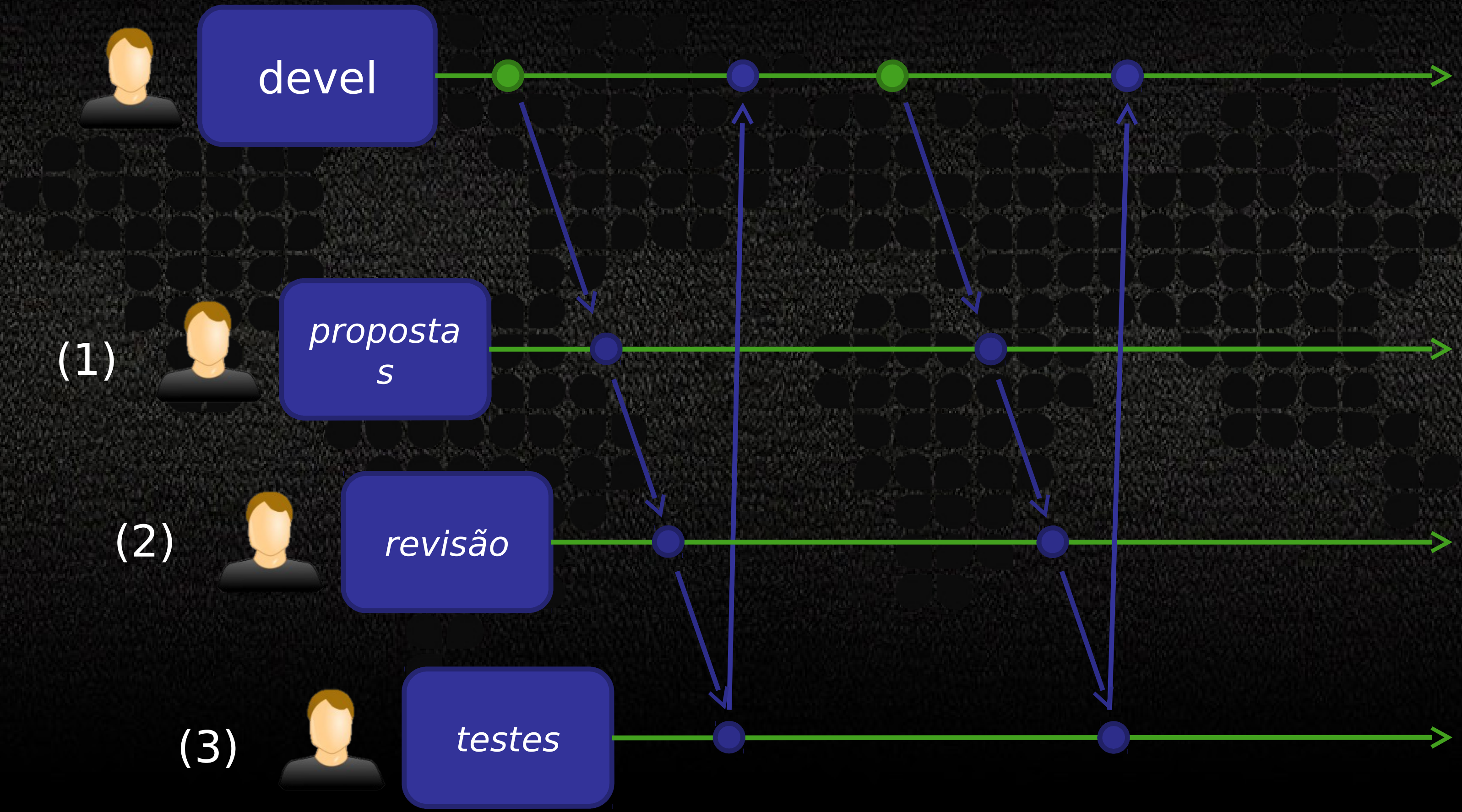


Version Control

Scenario:

Your development tasks need to progress in discrete levels of maturity: (1) change proposals, (2) analysis, (3) review, (4) unit tests, (5) integration tests, (6) system tests, etc.

Staged Integration Lines

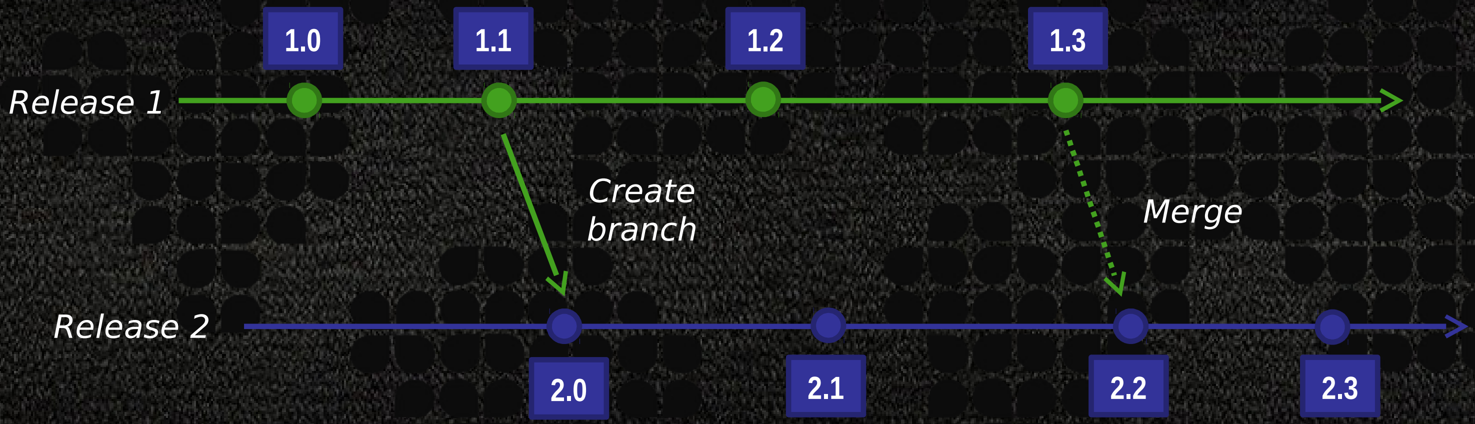


Some most common patterns.

When to create a branch?

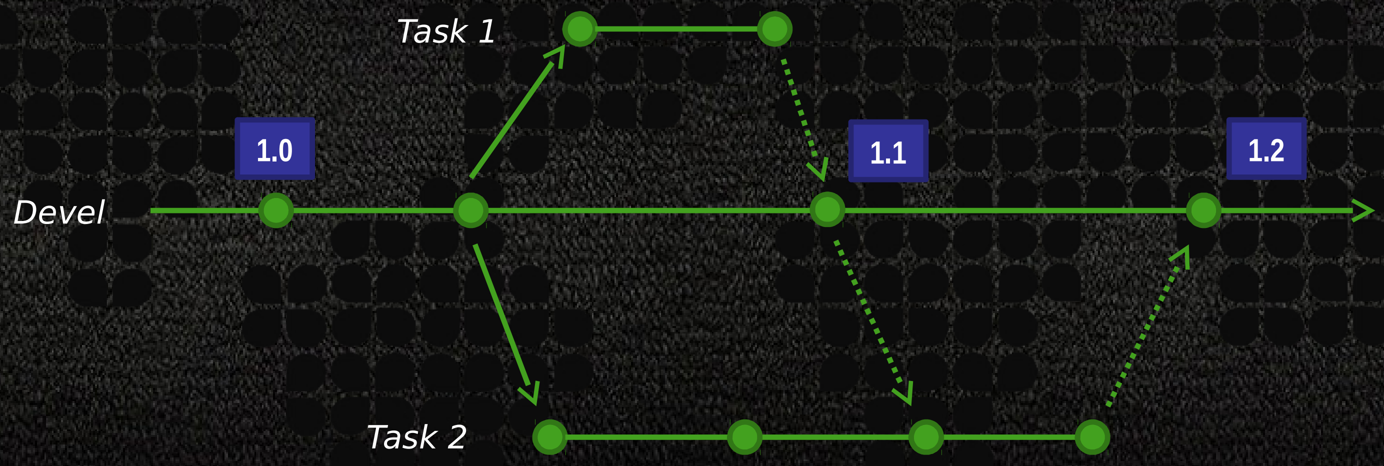
Version Control

Branch per Delivery



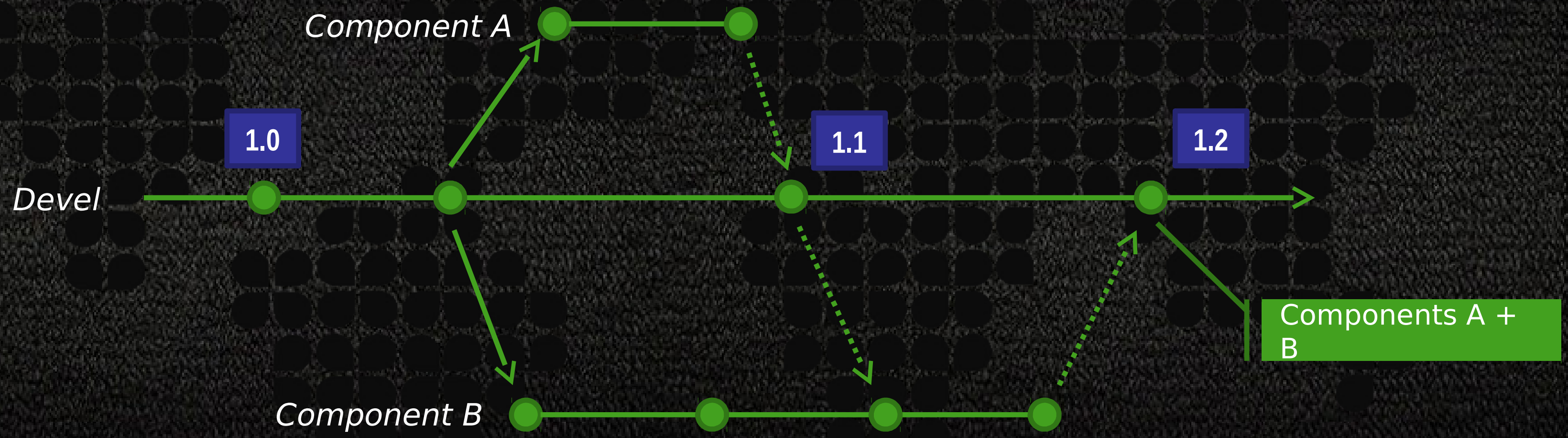
Version Control

Branch per Task



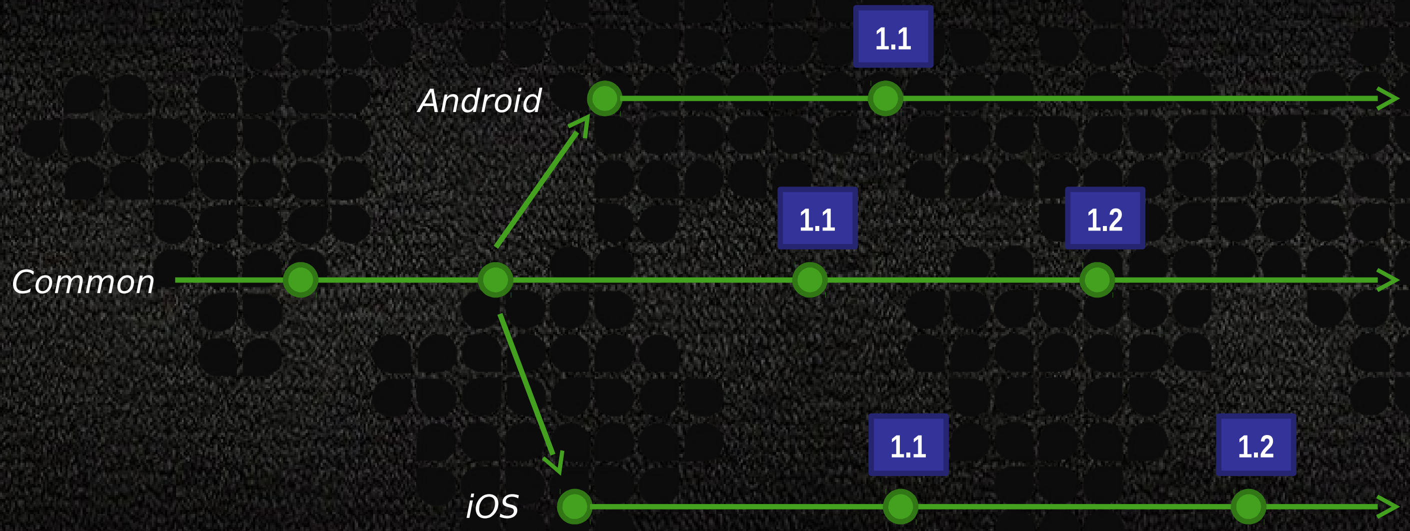
Version Control

Branch per Component



Version Control

Branch per Technology



What can happen if you don’t manage branching

Anti-Patterns

Anti-Pattern	Description
Merge Paranoia	avoiding merging at all cost, usually because of a fear of the consequences.
Merge Mania	spending too much time merging software assets instead of developing them.
Big-Bang Merge	deferring branch merging to the end of the development effort and attempting to merge all branches simultaneously.
Nerver-Ending Merge	continuous merging activity because there is always more to merge.
Wrong-Way Merge	merging a software asset version with an earlier version.
Branch Mania	creating many branches for no apparent reason.
Cascading Branches	branching but never merging back to the main line.
Mysterious Branches	branching for no apparent reason.
Temporary Branches	branching for changing reasons, so the branch becomes a permanent temporary workspace.
Volatile Branches	branching with unstable software assets shared by other branches or merged into another branch. **Note** Branches are volatile most of the time while they exist as independent branches. That is the point of having them. The difference is that you should not share or merge branches while they are in an unstable state.
Development Freeze	stopping all development activities while branching, merging, and building new base lines.
Berlin Wall	using branches to divide the development team members, instead of dividing the work they are performing.