Git

Alvin Berthelot

Version 1.0.0

Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons.

Attribution - Partage dans les Mêmes Conditions 3.0 non transposé.

La licence, ses explications ainsi que les moyens de contribution et réappropriation sont détaillés à la fin.

Introduction à Git

Tout est histoire

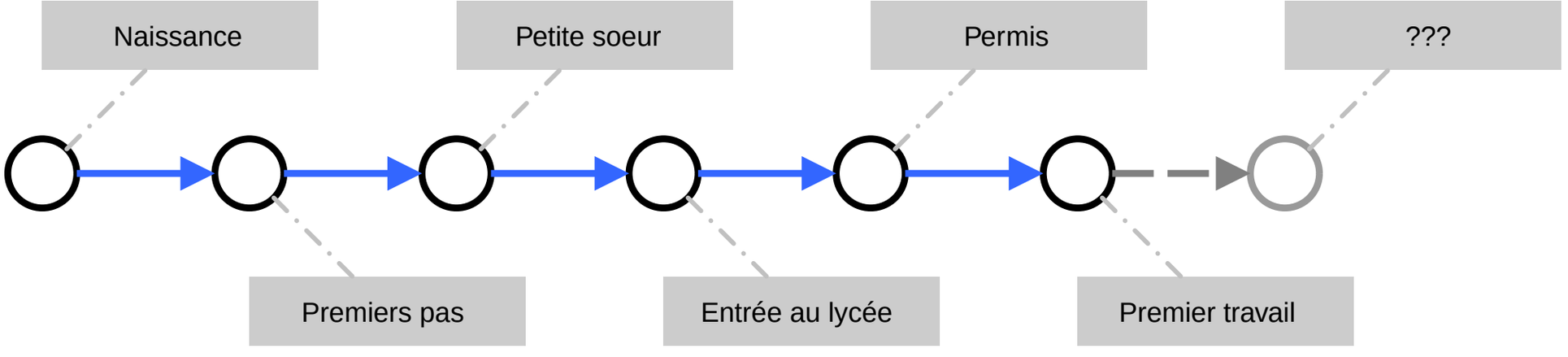
> L'histoire est « connaissance et récit des événements du passé, des faits relatifs à l'évolution de l'humanité (d'un groupe social, d'une activité humaine), qui sont dignes ou jugés dignes de mémoire ; les événements, les faits ainsi relatés ».
>
> — Petit Robert (2007)

C'est une très bonne définition qui peut s'appliquer à autres chose que l'humanité :

- Votre vie ou celle d'autres personnes
- Un livre, un film, une série
- Un logiciel informatique
- etc.

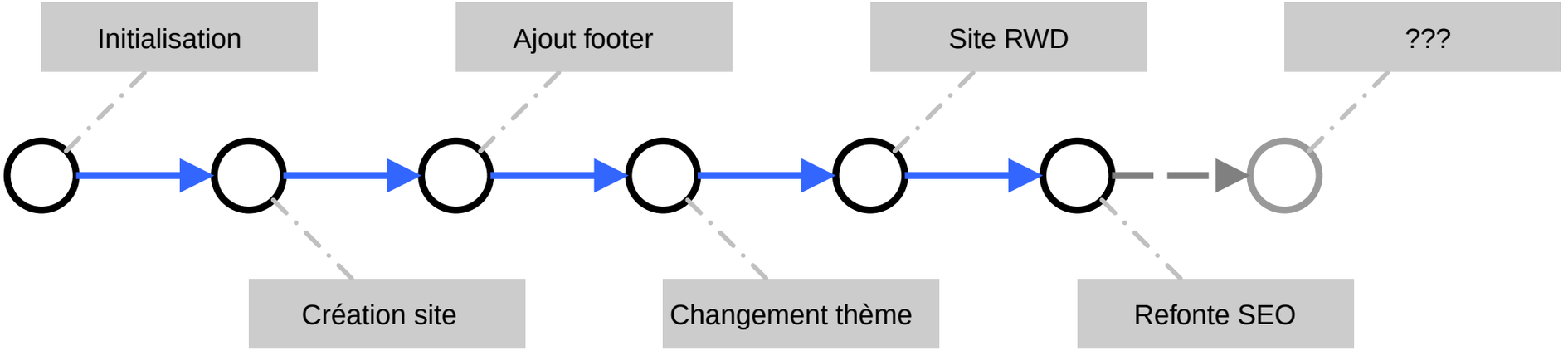
Une histoire en synthèse

- Événements
- Évolution
- Jugés dignes de mémoire



Une histoire pour un logiciel informatique

On peut très bien appliquer ces concepts au cycle de vie d'un logiciel informatique.



Outils de gestion de version

Pour répondre à ce besoin d'historiser un logiciel informatique, les **outils de gestion de version** sont apparus.

Ils permettent :

- De créer des "commits" qui sont la représentation des **événements**
- Suivre l'**évolution** en listant les "commits" avec un système d'horodatage
- Les "commits" sont **jugés dignes de mémoire** par les développeurs et cette responsabilité leur appartient

Les outils du marché

Ils existent plusieurs outils de gestion de version sur le marché, ils varient principalement selon :

- Le **mode** de fonctionnement (**centralisé** ou **décentralisé**)
- La **licence**
- L'**adoption** auprès de la communauté et au sein des entreprises

Les principaux

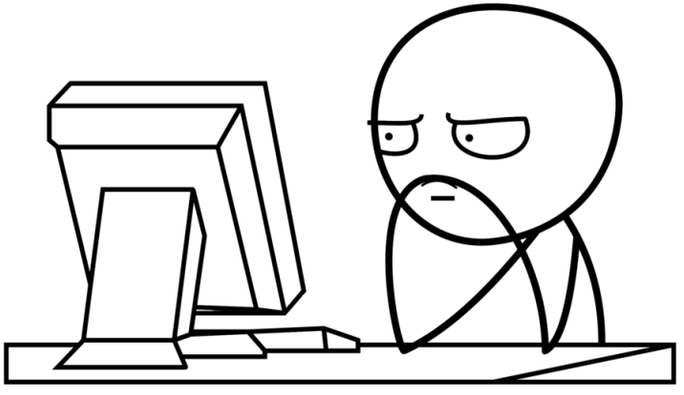
- **Git**
- Mercurial
- Subversion (SVN)
- ClearCase
- CVS

Git c'est quoi ?

Git est un **outil de gestion de version** qui fonctionne selon un **mode décentralisé**.

Il s'agit d'un **logiciel libre** créé en 2005 par Linus Torvalds.

Il est **très largement utilisé et maintenu par la communauté**.



Exercices

Notre premier objectif est d'installer Git sur sa machine, puis de s'assurer ensuite que le logiciel est opérationnel depuis une ligne de commande.

```
git --version
```

Certains éditeurs vous donnent accès aux fonctionnalités de Git par le biais de leur interface. **Il est primordial d'apprendre à utiliser Git avec la ligne de commande pour bien assimiler les concepts**.

Écrire une histoire avec Git

Initialiser une histoire avec Git

Initialiser une histoire pour un logiciel informatique correspond à observer un répertoire avec tout ce qui va s'y passer :

- **Ajout** d'un fichier au sein du répertoire
- **Modification** d'un fichier au sein du répertoire
- **Suppression** d'un fichier au sein du répertoire

Cette opération s'effectue avec la commande `git init`.

```
git init
```

Statut d'un dépôt Git

Il est possible de **savoir si un répertoire a déjà été initialisé avec Git** ou pas.

Comme il est possible de savoir si il y a eu des **ajouts, modifications ou suppressions de fichiers au sein du répertoire**.

Cette opération s'effectue avec la commande `git status`.

```
git status
```

Les différentes zones au sein du répertoire

Un répertoire suivi avec Git se décompose en 3 zones :

- La zone "copie de travail" (***working tree*** en anglais)
- La zone "en attente" (***staged*** en anglais)
- La zone "dépôt" (***repository*** en anglais)

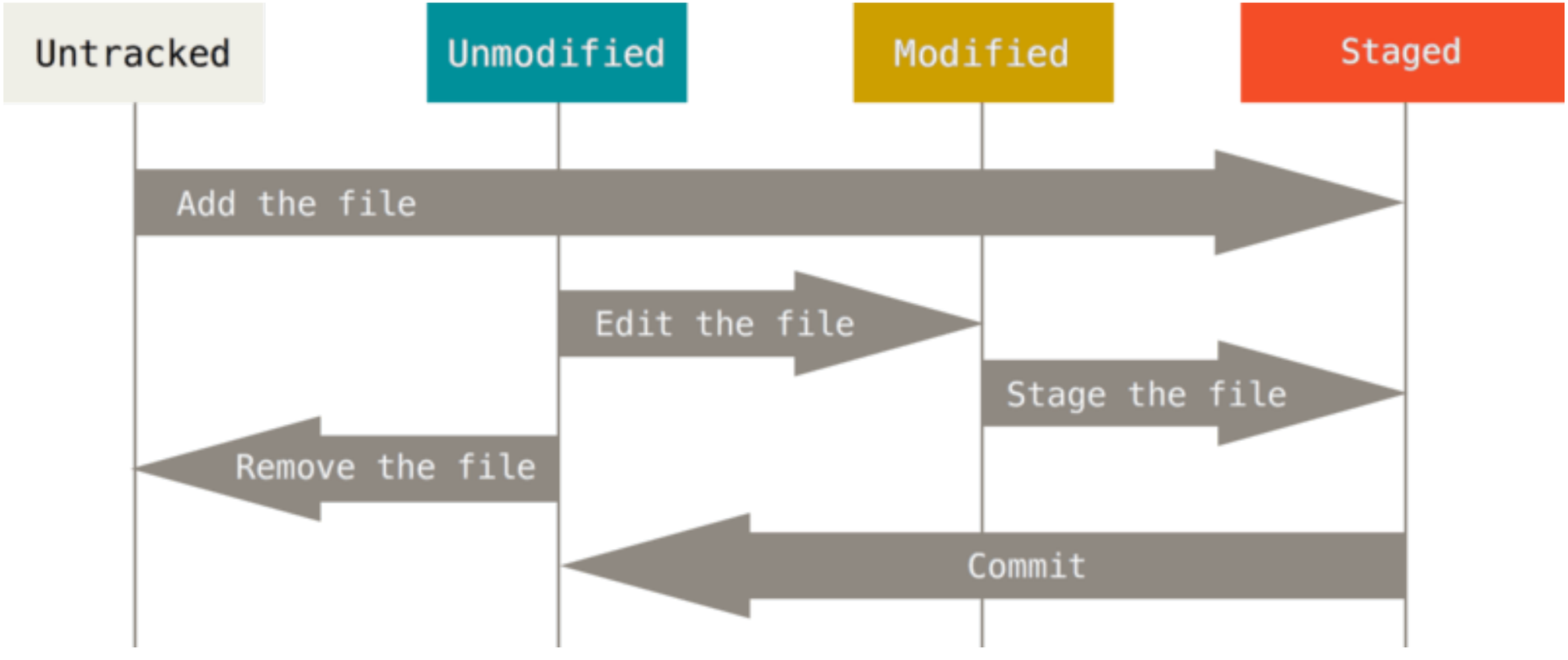
Les différents états des fichiers

La commande de statut fait apparaître les différents états des fichiers au sein du dépôt :

- Fichier non indexé / suivi (**untracked** en anglais)
- Fichier indexé mais non modifié (**unmodified** en anglais)
- Fichier indexé et modifié (**modified** en anglais)
- Fichier en attente de commit (**staged** en anglais)

Un même fichier peut avoir 2 états en simultanée dans certaines conditions.

Passer d'un état à un autre



Indéxer un fichier

Indéxer un fichier permet de le préparer pour un futur commit (une "mise en attente"), cette action est nécessaire si il est dans un état `untracked` ou `modified`.

Cette opération s'effectue avec la commande `git add` où il faut mentionner en argument le(s) fichier(s) à ajouter.

```
git add index.html ①

git add index.html main.css  ②

git add *.js ③

git add pictures/*  ④
```

① ajoute un unique fichier `index.html`

② ajoute les fichier `index.html` et `main.css`

③ ajoute tous les fichiers avec une extension `.js`

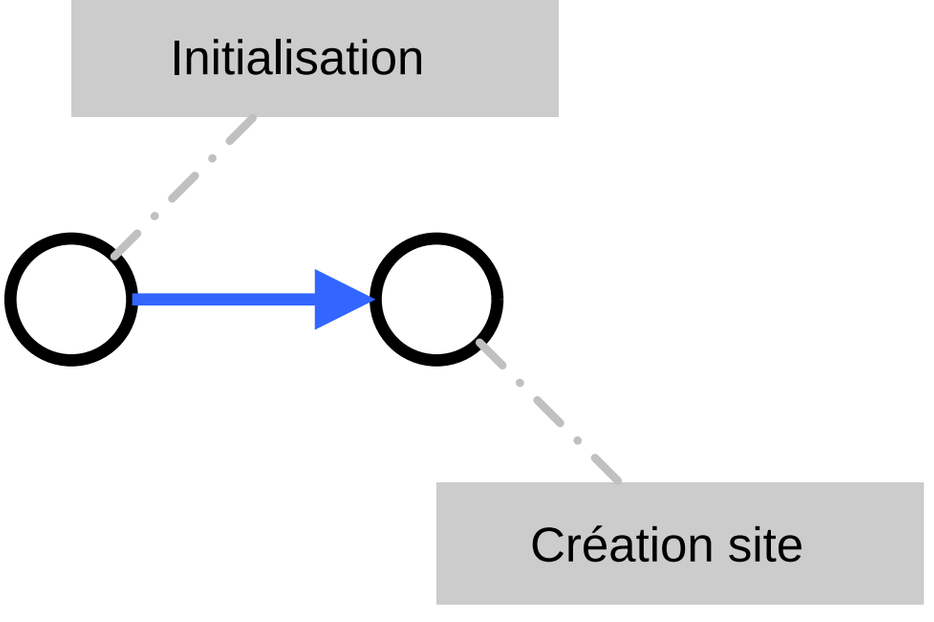
④ ajoute tous les fichiers se trouvant dant le répertoire `pictures`

"Commiter" un fichier

Une fois les fichiers préparés, il faut les "commiter" / enregistrer dans le dépôt.

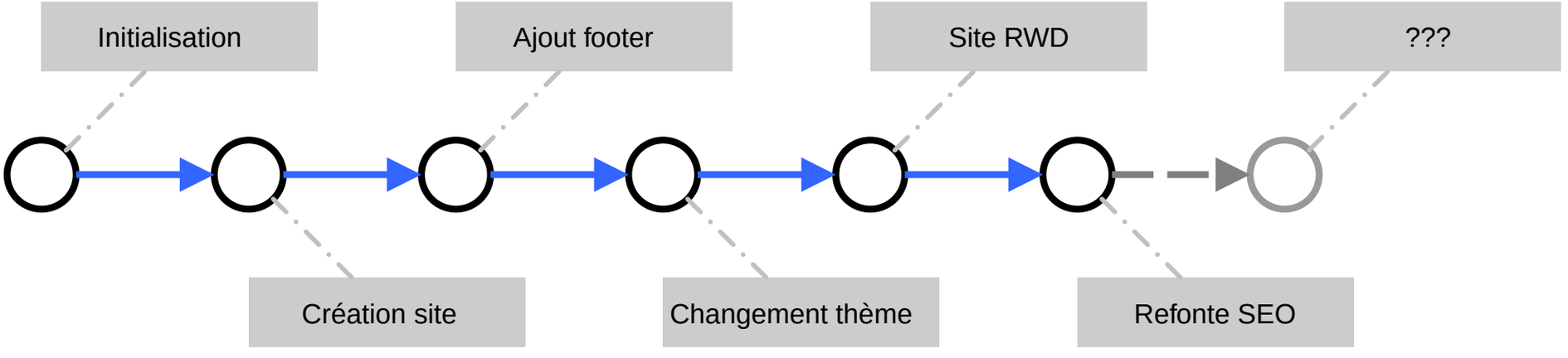
Cette opération s'effectue avec la commande `git commit` généralement suivi d'une option `-m` pour renseigner le message associé au commit.

```
git commit -m "Création site"
```



1 commit, 2 commits, 3 commits …

En continuant à ajouter, modifier et supprimer des fichiers et en commitant ces changements au fur et à mesure, on constribue à l'**évolution du logiciel**, on **construit son histoire**.



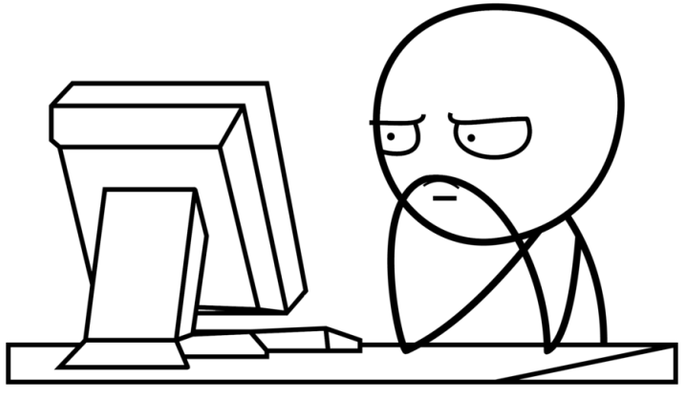
Lire une histoire avec Git

Une fois l'histoire écrite il est tout à fait possible de relire cette histoire avec tous les événements / commits qui s'y sont déroulés.

Cette opération s'effectue avec la commande `git log`.

```
git log
```

L'option `--oneline` devrait vous permettre d'y voir plus clair avec `git log`.



Exercices

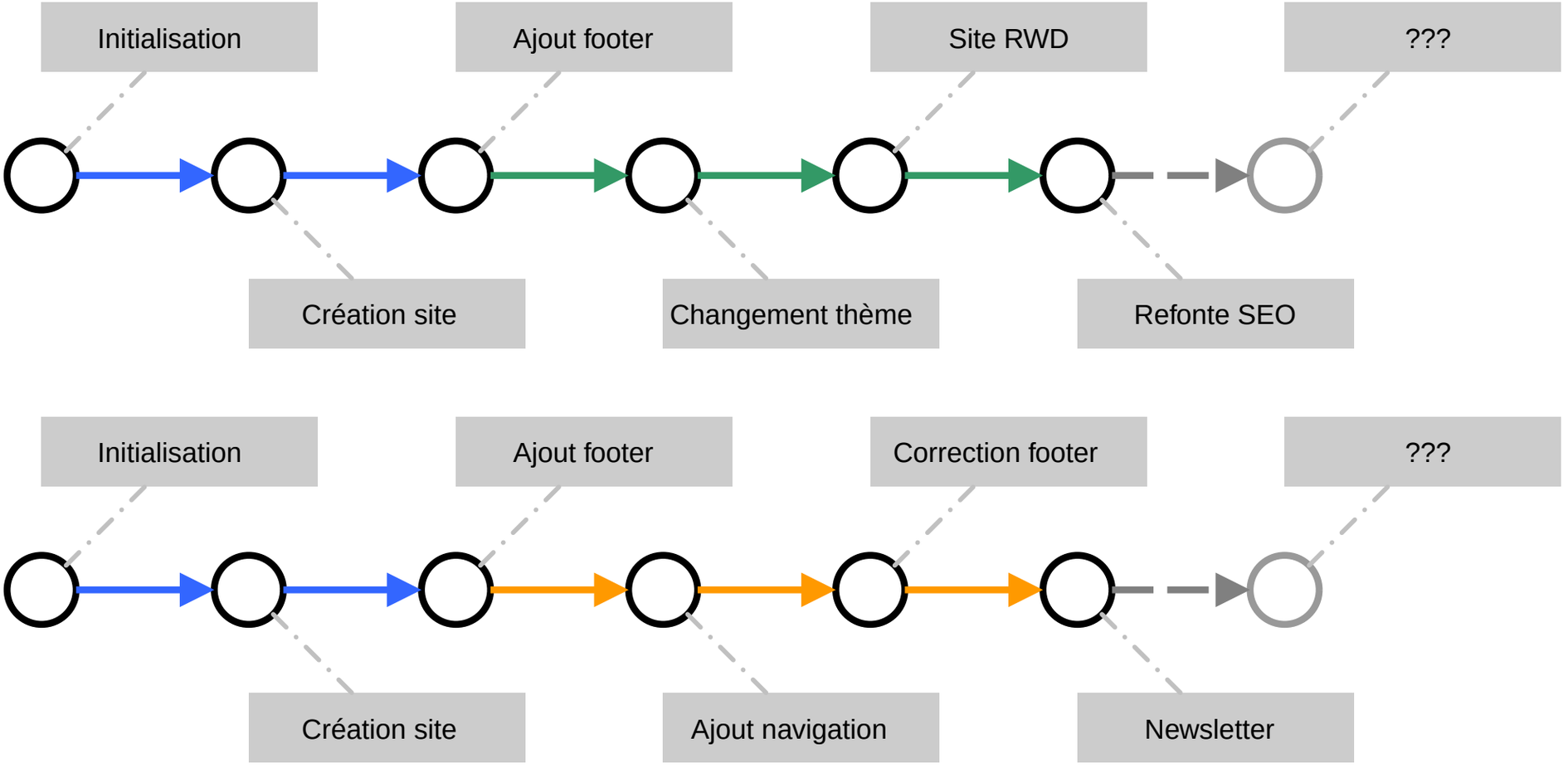
1. Vous devez créer un répertoire spécifique pour le site et l'initialiser avec Git
2. Assurez vous d'avoir un dépot Git avec `git status`.
3. Vous devez ajouter un fichier index.html avec un titre (h1) et un paragraphe (p).
4. Vous devez commiter le fichier créé
5. Vous devez ajouter un fichier main.css mettant en forme les paragraphes en bleu et relier ce ficher CSS au fichier HTML précédemment créé.
6. Vous devez commiter ces fichiers.

7. Vous devez ajouter un fichier main.js affichant "Hello world !" dans la console JavaScript et relier ce ficher CSS au fichier HTML précédemment créé.
8. Vous devez ajouter un style CSS pour mettre la balise h1 en rouge puis mettez le fichier concerné en attente pour le prochain commit.
9. Vous devez modifier le style CSS pour mettre les paragraphes en violet, puis utilisez la commande `git status`, que constatez vous ?
10. Vous devez afficher l'historique complet des commits.

Une histoire, une infinité d’alternatives

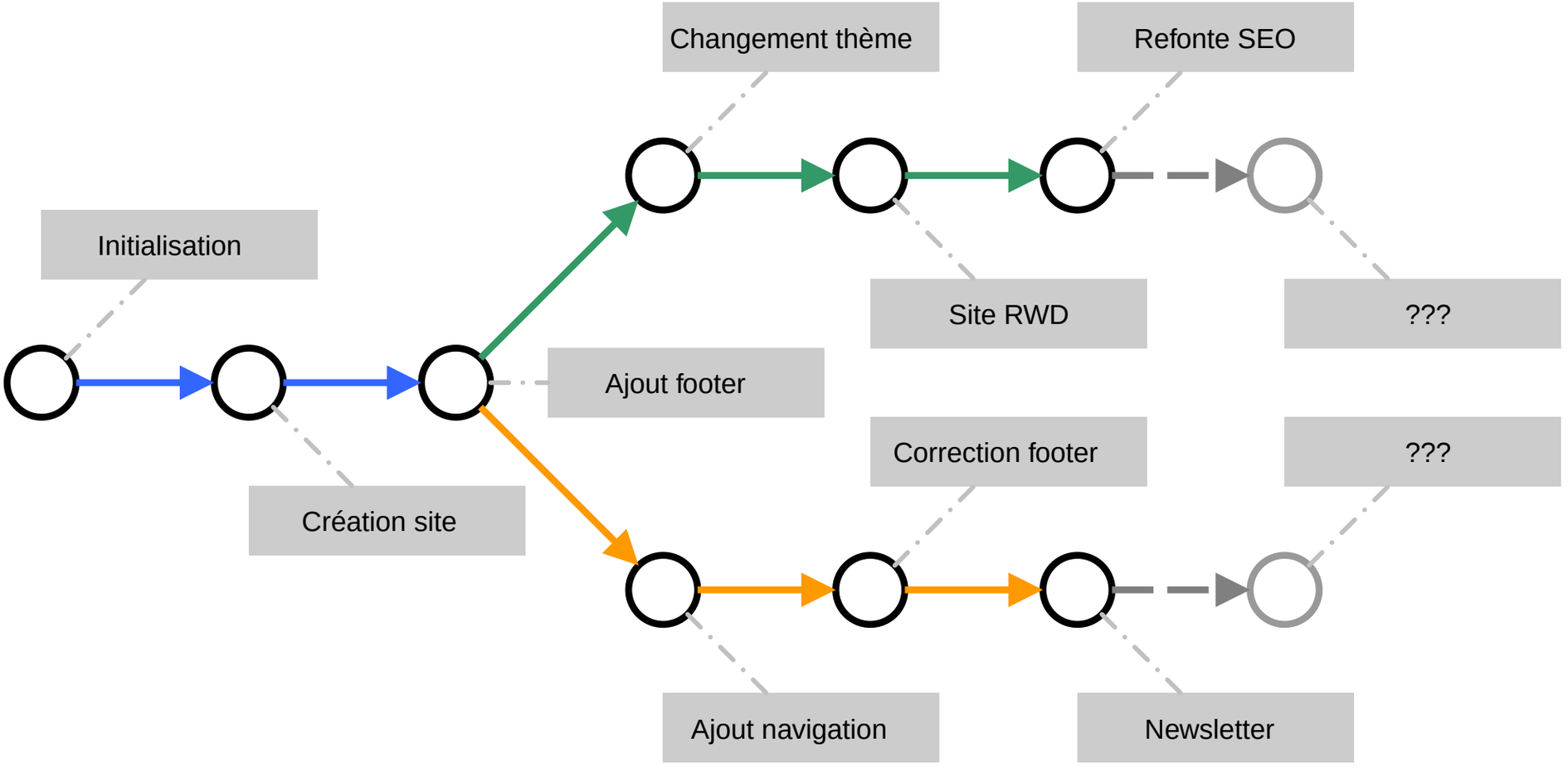
Univers parallèles

Une histoire à en soit un déroulement très linéaire, mais il est possible d'imaginer qu'elle aurait pu se dérouler autrement.



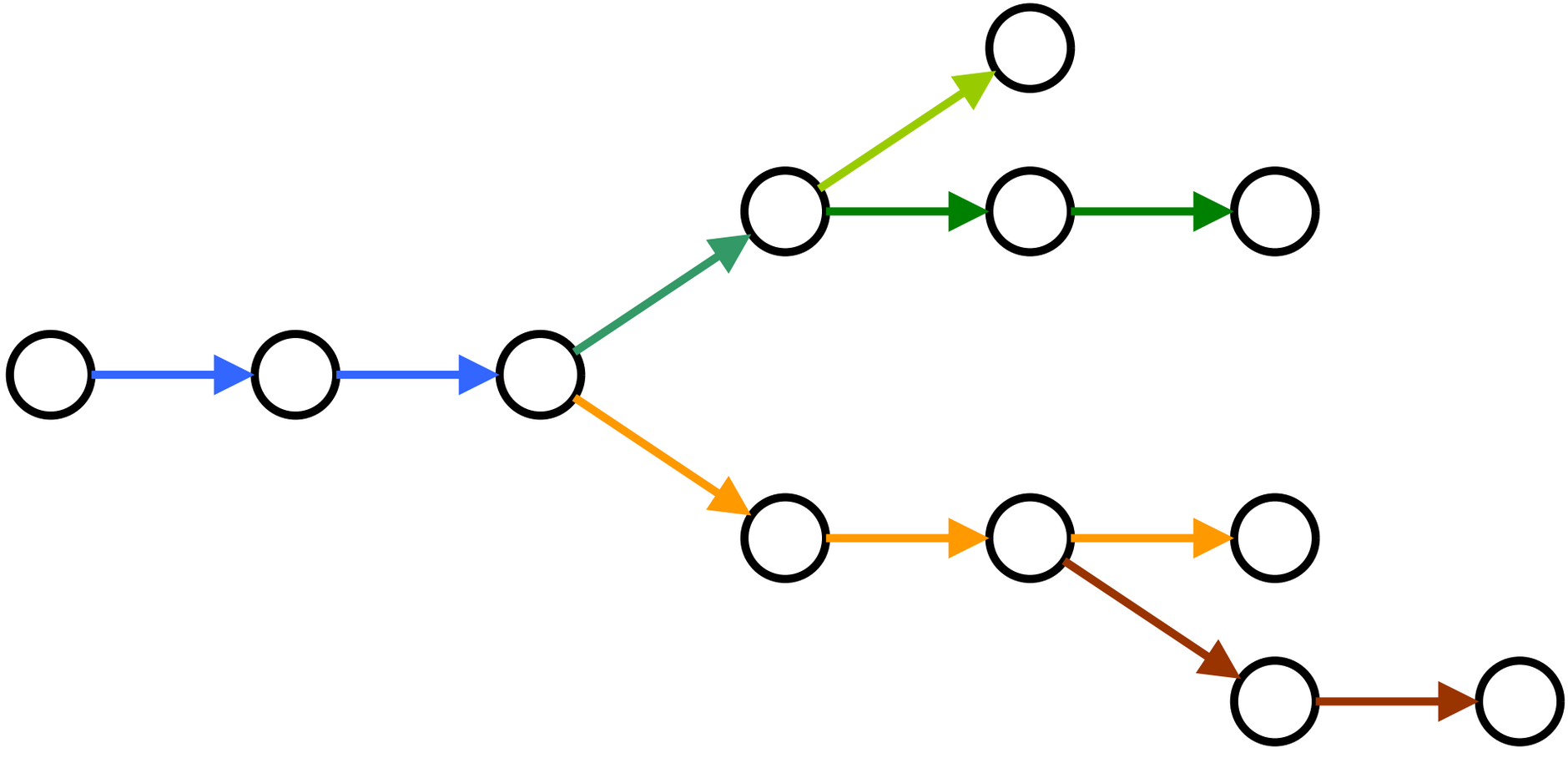
Tronc commun

La différence entre 2 alternatives d'une histoire et 2 histoires distinctes, c'est que **les alternatives ont forcément un tronc commun d'événements**.



Une infinité d'alternatives

Il est possible de créer **une infinité d'alternatives**, chacune pouvant démarrer à n'importe quel moment.



Les branches

Il est possible de créer ces alternatives pour un logiciel informatique avec les outils de gestion de version, on appelle cela **une branche**.

Cette opération s'effectue avec la commande `git branch` suivi du nom de la branche.

```
git branch rwd
```

Il faut d'ailleurs noter que par défaut on se situe déjà sur une branche, il s'agit de `master` c'est la branche par défaut.

Se positionner sur une branche

Pour écrire sur la nouvelle branche, **il faut se positionner dessus**.

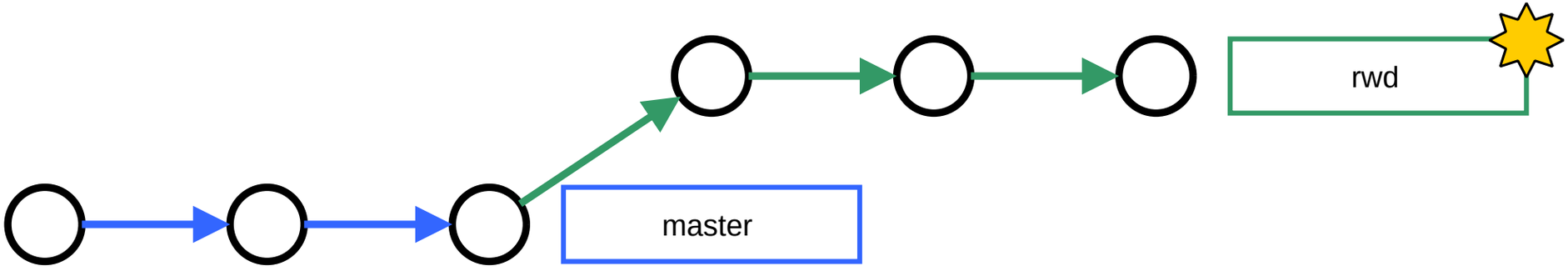
Cette opération s'effectue avec la commande `git checkout` suivi du nom de la branche.

```
git checkout rwd
```

Il est possible créer une branche et de se positionner directement dessus avec la commande `git checkout -b nom_de_la_branche`.

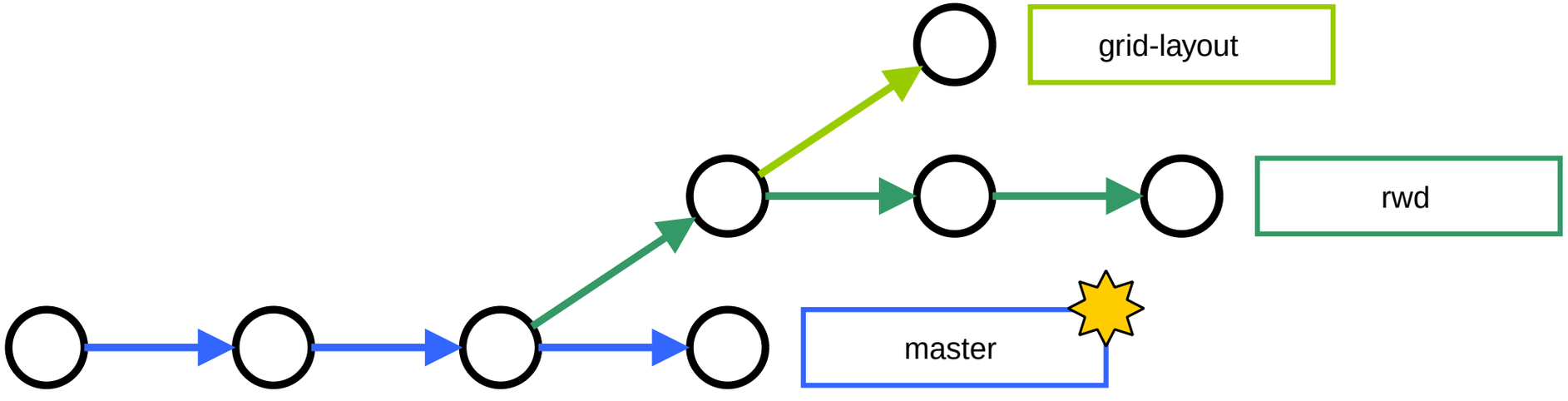
Continuer une branche

Il est alors possible d'écrire l'histoire de manière alternative en y ajoutant des commits.



Plusieurs branches en parallèle

On peut très bien continuer à construire plusieurs branches en parallèle.



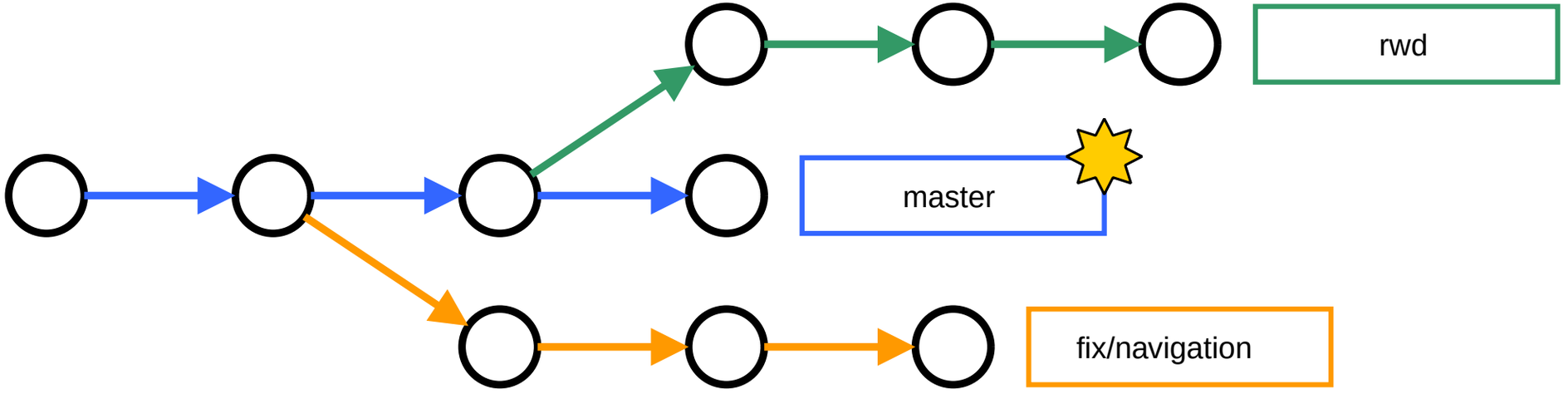
Donc finalement avec Git une histoire **n'est pas si linéaire** ...

Les branches quel intérêt ?

Les branches sont intéressantes car elles permettent d'avoir plusieurs alternatives en parallèle, cela permet de travailler sur des versions cloisonnées pour :

- Travailler sur une fonctionnalité
- Travailler sur un refactoring
- Ajouter une correction

D'autant que l'on peut créer une branche depuis un commit passé.



Se positionner sur un commit

Au même titre que l'on peut se positionner sur une branche, il est possible de se positionner sur un commit avec la commande `git checkout`.

Encore faut-il pouvoir indiquer sur quel commit on souhaite se positionner, 2 approches sont possibles :

- Récupérer l'identifiant unique du commit, son **SHA1**
- Utiliser un **tag** précédemment posé sur un commit

SHA1 kézako ?

Le contenu de chaque commit peut être représenté par un objet, chaque objet peut représenté par un identifiant unique via un algorithme de hachage, SHA1 en est un.

Chaque commit une fois réalisé peut donc être identifié par une chaîne de 40 caractères, on les retrouve facilement avec la commande `git log`.

On utilise généralement la version raccourcie du SHA1 (seulement 7 caractères), que l'on retrouve avec la commande `git log --oneline`.

Une fois le SHA1 du commit connu, il est possible de se positionner dessus.

Cette opération s'effectue avec la commande `git checkout` suivi du nom de la branche.

```
git checkout sha1_du_commit
```

Un tag kézako ?

En plus des SHA1, chaque commit peut être marqué une ou une plusieurs fois via des tags.

Les tags permettent de marquer explicitement un état ceci afin de retrouver facilement cet état.

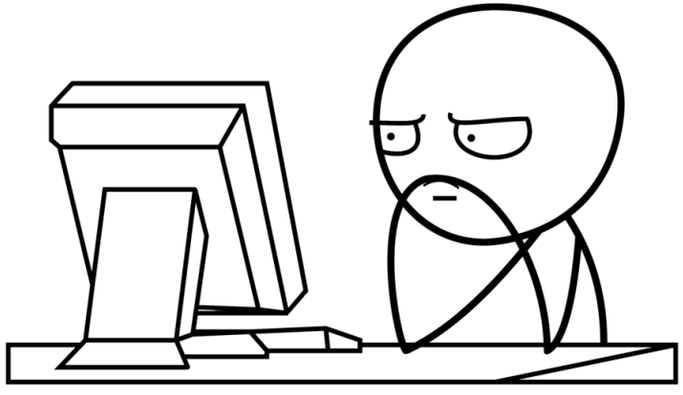
Cette opération s'effectue avec la commande `git tag` suivi du nom du tag.

```
git tag v1.4.8
```

Une fois le tag désiré connu, on se fiche un peu du commit concerné, on se positionne directement sur le tag.

Cette opération s'effectue avec la commande `git checkout` suivi du nom du tag.

```
git checkout v1.4.8
```



Exercices

Reprenez le répertoire utilisé précédemment.

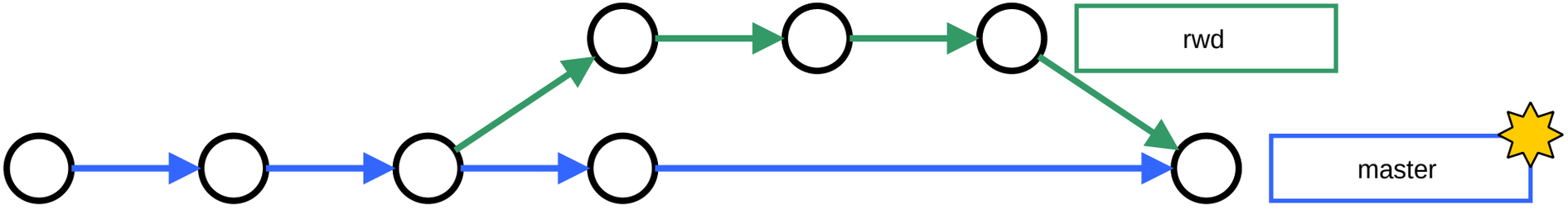
1. Vous devez ajouter un paragraphe mentionnant "Entreprise blobla" dans le fichier index.html.
2. Vous devez commiter cette modification. Imaginons que ce site est livré au client.
3. Vous devez créer une branche "footer" et positionnez vous dessus.
4. Vous devez ajouter un footer dans le fichier index.html, celui-ci doit contenir le paragraphe "Entreprise blabla, tous droits réservés".

5. Vous devez commiter cette modification.
6. Vous devez ajouter un style CSS pour mettre le footer en fond orange.
7. Vous devez commiter cette modification.
8. Repositionnez vous sur la branche "master".
9. Vous devez ajouter un header dans le fichier index.html, celui-ci doit contenir le mot "Navigation".
10. Vous devez commiter cette modification.
11. Vous vous apercevez d'une erreur, il aurait fallu écrire blabla et non blobla dans la version livrée au client. Positionnez vous sur la version livrée au client puis corrigez l'erreur.
12. Vous devez poser un tag "v1.0.0" sur la version corrigée.

Fusionner les alternatives

La fusion d'une branche

Dans le cas d'un logiciel informatique, créer une infinités d'alternatives est intéressant mais cela le devient encore plus lorsqu'on peut fusionner ces alternatives.



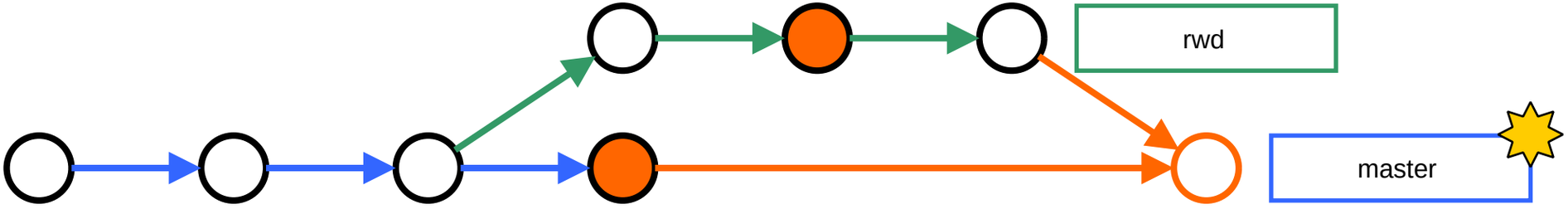
Cette opération s'effectue avec la commande `git merge` suivie du nom de la branche que l'on souhaite fusionner.

```
git merge rwd
```

Pour fusionner sur une branche, il faut être positionné sur cette branche.

Fusionner, pas toujours si simple

Git est assez intelligent pour fusionner directement des fichiers qui ont pu évoluer dans des branches si cela n'est pas conflictuel.



Si ça l'est, la fusion ne peut s'effectuer tant que les conflits n'ont pas été réglé.

Il n'y a pas de magie, l'arbitrage d'un conflit sur un fichier ne peut se faire que par une intervention humaine.

Régler un conflit, c'est quoi au juste ?

Régler un conflit consiste à créer des arrangements pour que 2 versions divergentes (A et B) d'un même fichier puissent à nouveau être compatibles.

Ces arrangements peuvent être de différentes sortes :

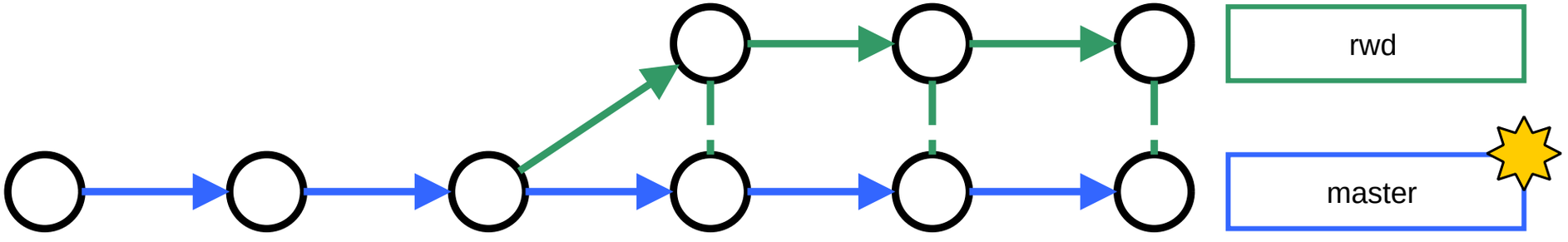
- La version A écrase la version B, ou inversement.
- Les versions A et B participent à la création d'une nouvelle version C.
- Une nouvelle version C totalement indépendante est créée.

Quel que soit le type d'arrangement choisi, cela génère nécessairement un commit "de merge".

Lors de l'arbitrage d'un conflit il faudra penser à indexer (`git add`) et commiter (`git commit`) les fichiers pour finir la fusion.

Le "Fast Forward"

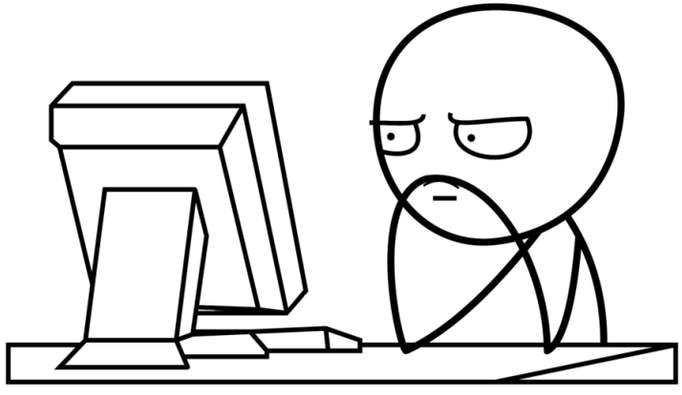
Si l'histoire est restée linéaire, le commit de merge est-il indispensable ?



Cette stratégie s'appelle le "Fast Forward" :

- **Avantage** : la lisibilité de l'histoire de la branche principale est plus directe.
- **Inconvénient** : on perd facilement la trace de la branche qui était à l'origine des commits.

> Il est possible de ne pas l'utiliser avec l'option `--no-ff`.



Exercices

1. Vous devez créer une branche "logjs" et positionnez vous dessus.
2. Vous devez modifier le fichier main.js pour qu'il affiche "Bonjour Git" désormais, puis committez.
3. Repositionnez vous sur la branche "master".
4. Vous devez modifier le fichier main.js pour qu'il ouvre une pop-up via alert(*Hello Git !*) désormais, puis committez.
5. Vous devez fusionner la branche logjs sur la branche master.
6. Vous devez créer une branche "stylelink" et positionnez vous dessus.

7. Vous devez modifier le fichier style.css pour que les liens (balise a) s'affichent en cyan, puis committez.
8. Vous devez modifier le fichier style.css pour que les liens (balise a) s'affichent en bleu au survol, puis committez.
9. Repositionnez vous sur la branche "master".
10. Vous devez modifier le fichier index.html pour que le libellé Navigation soit un lien redirigeant vers https://nantes.ynov.com/.
11. Vous devez fusionner la branche stylelink avec la branche master.
12. Vous devez créer une branche "stylelinkvisited" et positionnez vous dessus.
13. Vous devez modifier le fichier style.css pour que les liens (balise a) s'affichent en navy si ils ont déjà été visités, puis committez.
14. Repositionnez vous sur la branche "master".
15. Vous devez fusionner la branche stylelinkvisited avec la branche master.

Partager une histoire avec Git

Partager une histoire

Avoir une histoire et l'écrire est une chose primordiale.

Mais dans le cadre d'un logiciel, il est très rare que celle-ci soit la responsabilité d'une seule personne.

Très souvent un logiciel se construit avec l'aide de plusieurs personnes que ce soit à un instant t ou tout au long de la vie du projet.

Il faut donc être capable :

- De laisser d'autres personnes récupérer l'histoire (la même pour tous)
- De laisser d'autres personnes participer à l'histoire

Il faut donc **centraliser** l'histoire. Git permet de centraliser une histoire mais de façon décentralisée.

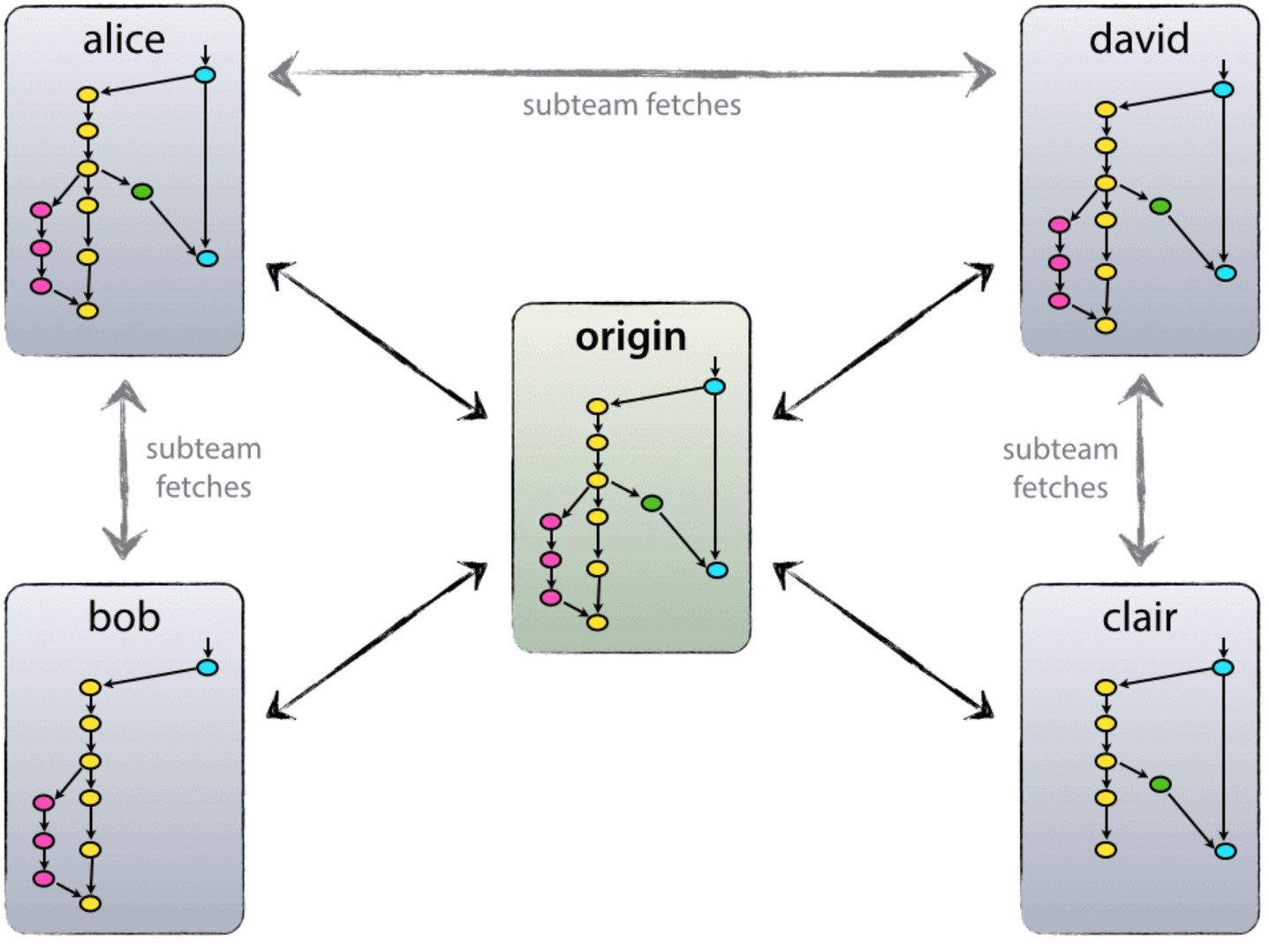
Centralisé Vs Décentralisé

Un dépôt centralisé ne permet pas de récupérer une version des fichiers à un instant t, contrairement au décentralisé qui récupère l'ensemble du dépôt.

Cela à de nombreux avatages :

- Un dépôt décentralisé n'est pas un SPOF
- On peut travailler sur un dépôt (avec toute son histoire) sans avoir de connexion
- Gain de performance concernant la latence réseau
- On peut peaufiner un dépôt local avant de le partager

Décentralisé, oui mais ...



Récupérer une histoire

Git permet de récupérer (ou cloner) une histoire avec la commande `git clone` suivie de l'adresse du dépôt (sous forme d'URL).

```
git clone https://github.com/alvinberthelot/git_2018_B1
```

Le dépôt est accessible si son accès en lecture est publique, un dépôt peut très bien être masqué en privé.

Avec la commande `clone` on récupère les sources comme dépôt local mais on associe directement un dépôt distant qui par défaut s'appelle "origin".

Les dépôts distants

Un dépôt local peut avoir plusieurs dépôts distants mais également aucun selon la manière dont le dépôt local a été initialisé.

- La commande `git init` initialise un dépôt local mais pas de dépot distant
- La commande `git clone` initialise un dépôt local en fonction d'un dépot distant, ce dépôt distant est par convention nommé "origin"
- Il est possible d'ajouter d'autres dépôts distants avec la commande `git remote add`

La commande `git remote` permet de connaître les dépots distants liés à un dépôt local.

```
git remote
```

L'option `-v` permet de connaître l'URL des dépôts distants.

Ajouter un dépôt distant

La commande `git remote add` suivie d'un nom et d'une URL permet d'ajouter un dépôt distant avec un nom précis.

```
git remote add alvin https://github.com/alvinberthelot/git_2018_B1
```

Se synchroniser avec un dépôt distant

La commande `git fetch` suivie du nom du dépôt et de la branche désirée permet de récupérer les fichiers **sans fusion**.

```
git fetch origin master
```

La commande `git pull` suivie du nom du dépôt et de la branche désirée permet de récupérer les fichiers **avec fusion**.

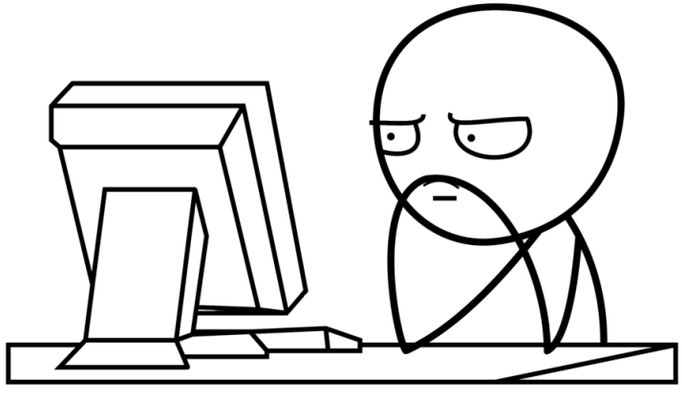
```
git pull origin master
```

Contribuer à un dépôt distant

La commande `git push` suivie du nom du dépôt et de la branche désirée permet d'envoyer les fichiers.

```
git push origin master
```

Si la lecture d'un dépôt est publique, ce n'est pas la même histoire concernant l'écriture.



Exercices

Dépot 1

1. Vous devez créer un nouveau dépôt vide sur votre compte GitHub.
2. Vous devez ajouter ce dépôt distant à votre dépôt local.
3. Vous devez "pousser" votre code de votre dépôt local sur votre nouveau dépôt distant.

Dépot 2

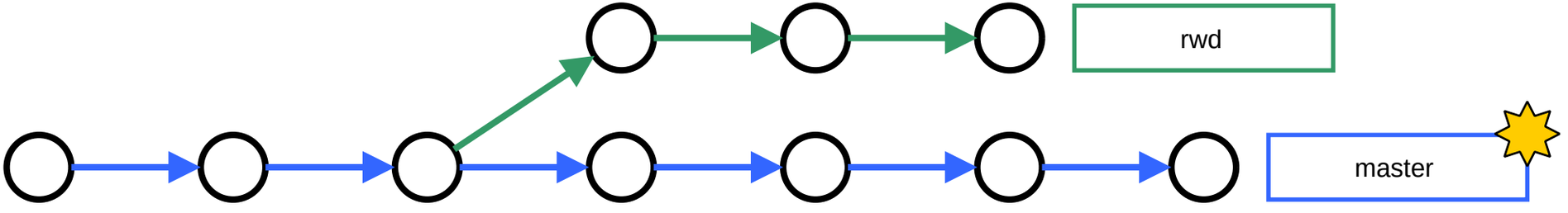
1. Vous devez récupérer votre dépôt distant git_2018_B1 en local.
2. Vous devez lister les dépôts distants.
3. Vous devez ajouter un dépôt distant "alvin" avec l'URL https://github.com/alvinberthelot/git_2018_B1.
4. Vous devez lister les dépôts distants.
5. Vous devez créer une branche "tomato" et positionnez vous dessus.
6. Vous devez modifier le fichier style.css pour que les liens (balise a) s'affichent en "tomato" au survol, puis committez.
7. Vous devez "pousser" votre code de votre dépôt local sur votre dépôt distant sur la branche tomato.
8. Vous devez "pousser" votre code de votre dépôt local sur le dépôt distant "alvin" sur la branche tomato. Que constatez vous ?
9. Vous devez vous repositonnez sur la branche master.

10. Vous devez fusionner la branche tomato avec la branche master.
11. Vous devez récupérer les dernières modifications de la branche master du dépôt distant et les fusionner.

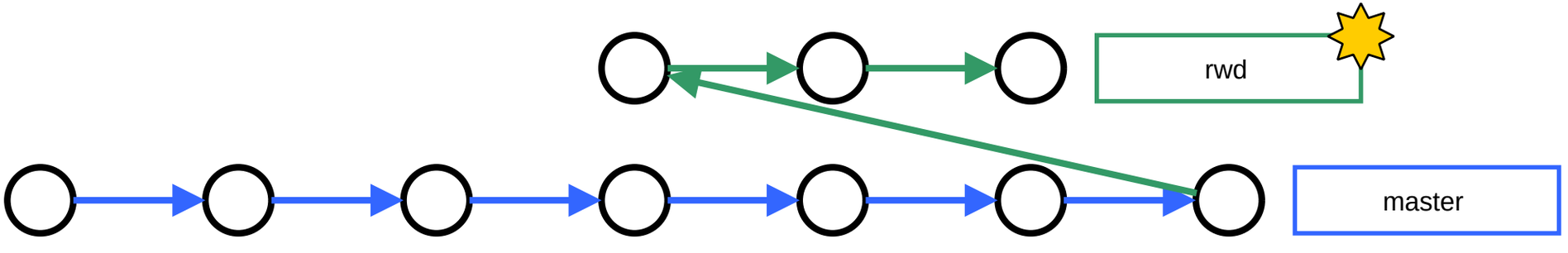
Réécrire l'histoire

Revoir le tronc commun

Si une branche a été tirée à un instant t sur un commit, elle aurait très bien pu être tirée avant ou après.



On cherche généralement à faire (re)partir une branche d'un commit plus tardif pour que celle-ci puisse profiter des avancées.



Disposer d'un nouvelle base

Cette opération s'effectue avec la commande `git rebase` suivie du nom de la branche d'où l'on souhaite (re)partir et du nom de la branche qui va en profiter.

```
git rebase master rwd
```

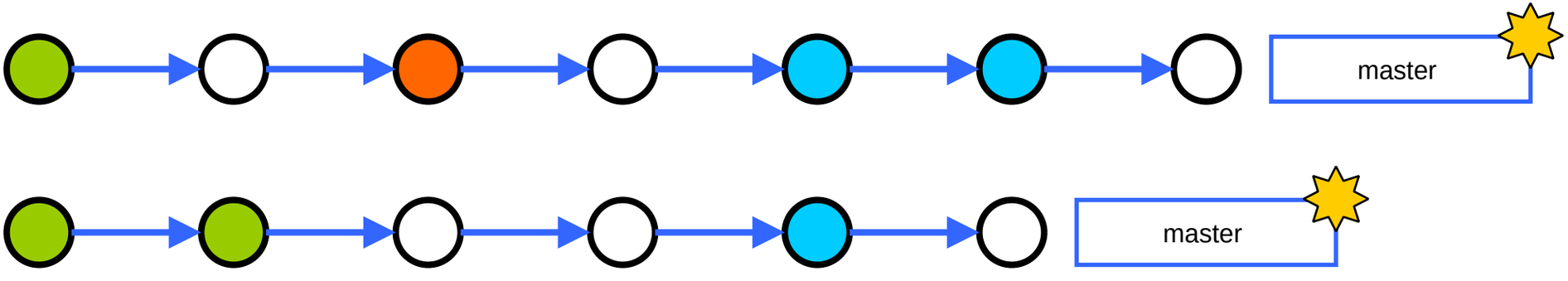
L'opération rejoue les commits de la branche qui va en profiter à partir de la nouvelle base.

Cette opération peut apporter son lot de conflits au même titre qu'un `merge`, mais ce n'est pas la même opération.

Changer les événements

Si il est possible de rejouer des commits sur une autre branche, est-ce possible sur une même branche ?

C'est le cas et cela permet de changer les événements qui ont eu lieu sur une même branche.



Cette opération s'effectue avec la commande `git rebase -i` suivie du SHA1 du commit qui va servir de point de départ.

```
git rebase -i 4bf582d
```

Que peut-on changer ?

Dans un rebase interactif plusieurs options sont possibles pour chaque commit :

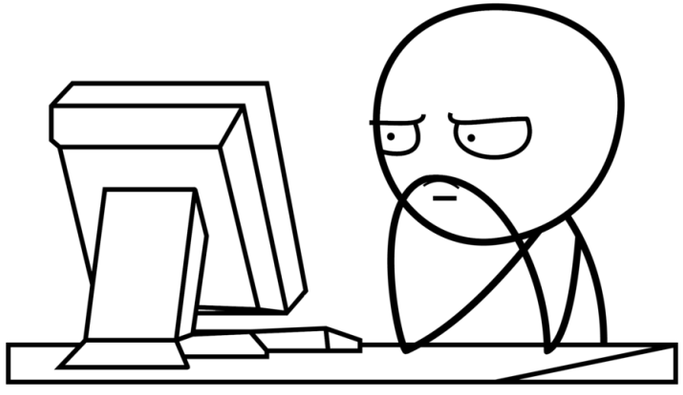
- pick : Rejouer un commit tel quel.
- reword : Changer le message.
- edit : Préparer le commit mais laisser la possibilité à des changements.
- squash : Fusionner un commit avec le commit précédent.
- fixup : Fusionner un commit avec le commit précédent, sans garder la trace du message.
- exec : Exécuter une commande.
- drop : Supprimer un commit.

Le commit qui va servir de point de départ ne va être pris en compte.

Les conséquences

Changer l'histoire c'est super puissant mais ça peut être très dangereux.

Si vous changez une histoire qui a déjà été poussé sur un dépôt distant et que vous l'écrasez vous allez désynchroniser tous les autres …



Exercices

1. Si ce n'est déjà fait vous devez ajouter un dépôt distant "alvin" avec l'URL https://github.com/alvinberthelot/git_2018_B1.
2. Vous devez synchroniser votre master avec la branche master du dépôt distant.
3. Vous devez créer une branche "participation" et positionnez vous dessus.
4. Vous devez modifier le fichier index.html pour qu'il affiche votre nom dans la liste d'étudiants. Vous devez commiter ce fichier.
5. Profitez pour customiser le style CSS pour votre nom. Faites cette opération

avec plein de commits.

6. Si vous faisiez le ménage un peu dans vos commits.
7. Il a fait quoi le prof pendant ce temps là ?. Resynchroniser votre master avec son dépôt.
8. Refaites partir votre branche "partcipation" depuis cette nouvelle base.
9. Bonus faites une "pull request" depuis GitHub.

Licence

Pour toutes questions, réclamations ou remarques, merci d'envoyer un message à alvin.berthelot@webyousoon.com.

Explications licence CC BY-SA 3.0

Cette licence permet aux autres de remixer, arranger, et adapter votre œuvre, même à des fins commerciales, tant qu'on vous accorde le mérite en citant votre nom et qu'on diffuse les nouvelles créations selon des conditions identiques.

Cette licence est souvent comparée aux licences de logiciels libres, "open source" ou "copyleft".

Toutes les nouvelles œuvres basées sur les vôtres auront la même licence, et toute œuvre dérivée pourra être utilisée même à des fins commerciales.

C'est la licence utilisée par Wikipédia ; elle est recommandée pour des œuvres qui pourraient bénéficier de l'incorporation de contenu depuis Wikipédia et d'autres projets sous licence similaire.

Contribution et réappropriation

Ce fichier PDF est généré avec Asciidoctor à partir d'un dépôt Git se trouvant sous GitHub.

https://github.com/alvinberthelot/slides-git

Cela signifie que vous n'avez pas besoin de vous battre avec un fichier binaire (le PDF) pour **contribuer**, **vous réapproprier le contenu** ou **modifier le thème** de présentation.

Contribution

Vous voulez **contribuer au contenu** car :

- Il y a une erreur (ça arrive à tout le monde), de typographie, de compréhension, ou tout autre chose.
- Vous souhaitez apporter une précision.

Il vous suffit de contribuer au projet via Git par le moyen d'une "pull request" sur le dépôt Git.

Réappropriation

N'oubliez pas les conditions de la licence.

Vous voulez vous **réapproprier le contenu** car :

- Vous souhaitez donner un style différent.
- Vous souhaitez enlever/ajouter/modifier des sections dans votre contexte.

Il vous suffit de "forker" le dépôt Git et d'y apporter vos propres modifications, puis de générer par vous même le nouveau PDF.