<title> Vim avec une tasse de café </title> <script defer data-domain="vim.avec.une-tasse-de.cafe" src="https://stats.192168128.xyz/js/plausible.js"></script>

CyberPrez #03 - VIM

Découvrir Vim

Sommaire

(1/2)

Sommaire

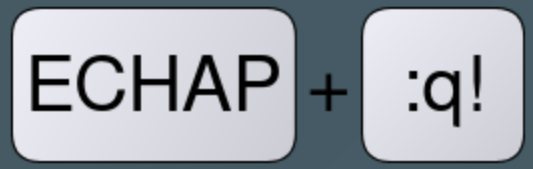
(2/2)

Qu'est-ce que Vim ?

Vim est un utilitaire en ligne de commande permettant de modifier des fichiers texte. Celui-ci est un dérivé de l'éditeur `vi`, qui est intégré dans la plupart des systèmes Unix-like.

Vim est connu pour être complexe à prendre en main, mais une fois que l'on a compris les bases, il est très puissant et permet de gagner beaucoup de temps.

Si vous avez trouvé cette présentation parce que **vous êtes bloqué dans vim** depuis plusieurs heures : *Respirez un bon coup*, pressez la touche `echap` et tapez `:q!` pour quitter sans sauvegarder.



Si vous voulez sauvegarder avant de quitter le document, pressez la touche `echap`, puis `:wq`.

Et maintenant que j'ai perdu 90% de mon audience, je peux commencer la réelle présentation.

... mais avant ça, un peu d'histoire.

Histoire de vim

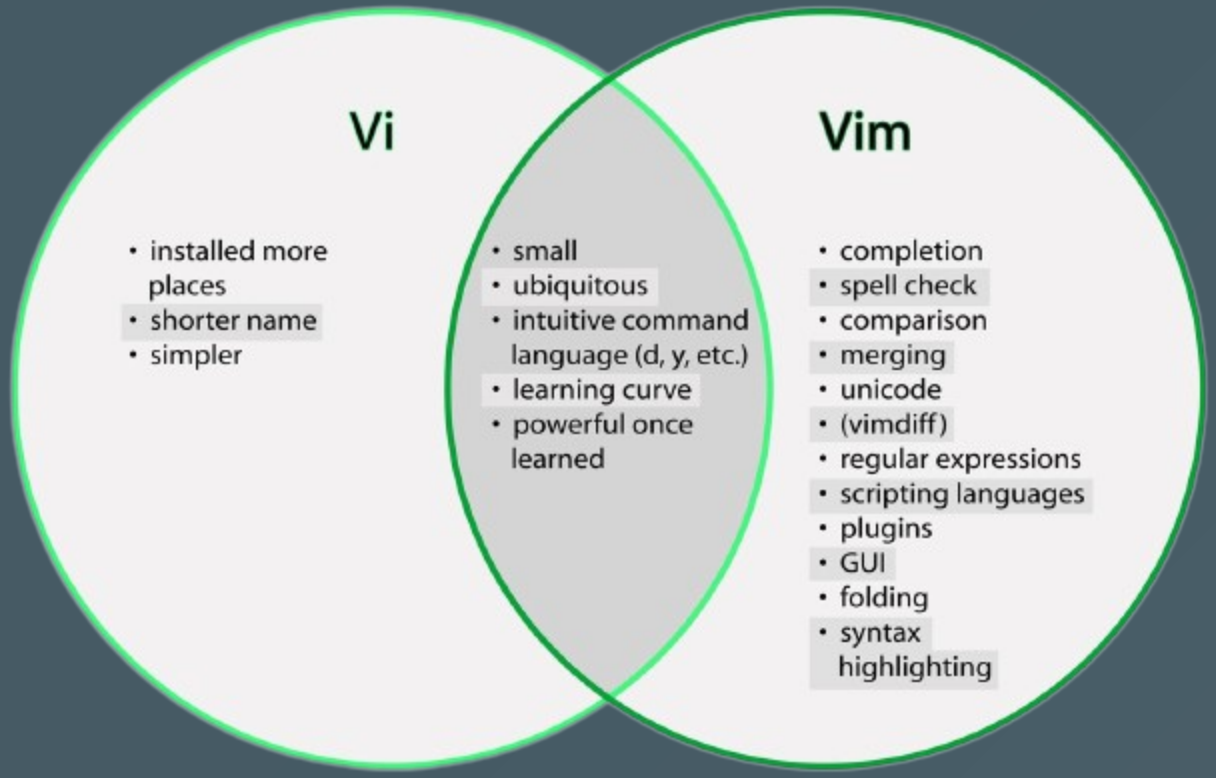
Vim, qui signifie **"Vi Improved"** *(Vi amélioré)*, est l'éditeur de texte le plus utilisé en ligne de commande. Son histoire remonte aux années 1970 avec l'apparition d'un éditeur de texte appelé Vi *(Visual Editor)* développé par *Bill Joy* pour le système d'exploitation Unix.

Vi est devenu populaire parmi les programmeurs et les administrateurs système grâce à sa puissance et à sa simplicité d'utilisation en mode texte. Au fil des années, de nombreux utilisateurs ont contribué au développement de Vi en créant des versions personnalisées, chacune avec ses améliorations et fonctionnalités spécifiques.

En *1991*, *Bram Moolenaar* a créé Vim en se basant sur le code source de Vi. Vim était destiné à être une version améliorée de Vi, incorporant de nouvelles fonctionnalités et offrant une plus grande flexibilité.

 C'est lui

Parmi les améliorations, on peut retrouver la coloration syntaxique, l'édition multifenêtres, le support des plugins et la possibilité d'étendre les fonctionnalités grâce à un langage de script interne.



Bref, il a repris la philosophie et la base de <code>vi</code>, et il en a fait un editeur incroyablement plus agréable et performant.

Aujourd'hui, Vim continue d'évoluer grâce à une communauté active qui propose des mises à jour, des corrections de bugs et des fonctionnalités supplémentaires.

Il reste un outil essentiel pour les développeurs et les utilisateurs expérimentés qui apprécient sa puissance et sa capacité à optimiser leur flux de travail.

L'intégralité des donations vers Vim sont redirigées vers l'ICCF Holand pour aider des enfants en Ouganda. Vous pouvez voter pour les futures fonctionnalitées de Vim après une donation.

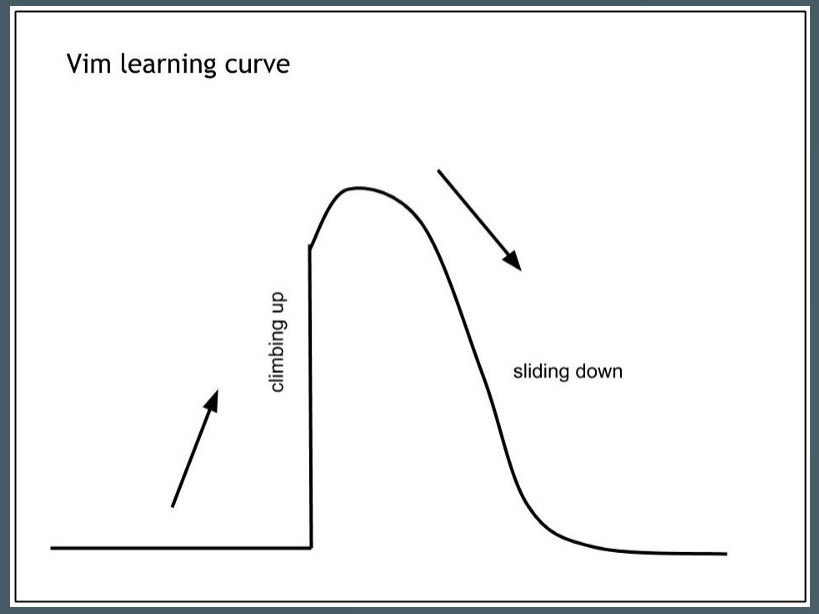
Maintenant que le cours d'histoire est terminé, passons à la pratique.

La base de Vim

La base de Vim

N'oubliez pas que VIM s'apprend avant-tout par la **pratique**. Il est donc conseillé de suivre cette présentation en même temps que vous lisez les slides.

Ne désespérez pas. Vous développerez des automatismes au fur et à mesure que vous l'utiliserez.



Petit conseil amical avant de commencer

```
alias nano="vim"
```

Je vous assure que vous ne le regretterez pas.

 Et rendez cet alias persistant dans votre <code>.bashrc</code>

Les modes

La première chose à savoir sur Vim, c'est qu'il existe plusieurs modes avec lesquels vous allez régulièrement permuter.

Chaque mode possède un rôle différent. Il est important de comprendre comment fonctionnent les modes pour pouvoir utiliser Vim correctement.

Nous utiliserons principalement 3 modes : le mode **normal**, le mode **insertion** et le mode **visuel**.

- Le mode **normal** est celui par défaut, il permet de lancer des 'macros' et des 'commandes'.
- Le mode **insertion** peut se lancer en pressant '`i`' afin d'éditer un texte sans raccourcis.
- Le mode **visuel** s'active en appuyant sur '`v`', il permet de modifier une sélection.

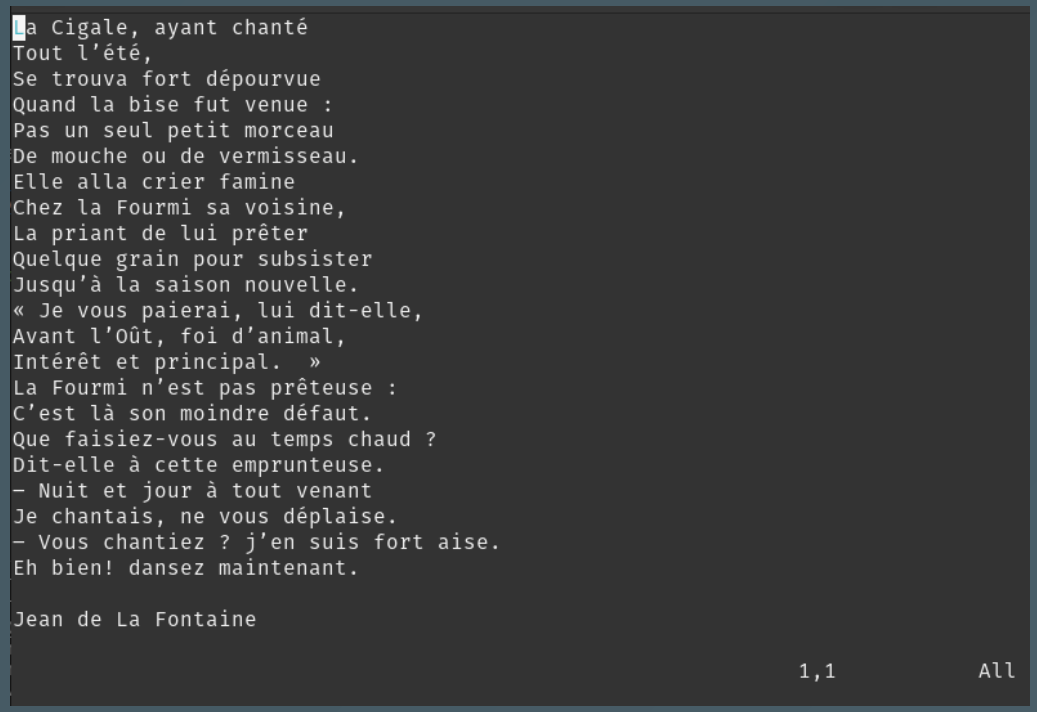
Le mode dans lequel nous nous trouvons est indiqué en bas à gauche de l'écran. Si vous êtes en mode **insertion**, vous devriez voir `-- INSERT --` en bas à gauche de l'écran.

Le mode **visuel** sera affiché de la même manière, avec `-- VISUAL --`.

Vous pouvez **toujours** revenir au mode `normal` en pressant la touche `echap`.

Téléchargez maintenant le fichier exemple `cigale-et-la-fourmi.txt` ici.

Pour l'ouvrir avec Vim, tapez `vim cigale-et-la-fourmi.txt` dans votre terminal ou tapez `vim` puis écrivez `:edit cigale-et-la-fourmi.txt`.



Apprenons à se déplacer dans un fichier

À l'origine, les touches directionnelles n'existaient pas. Pour se déplacer dans le texte, il fallait utiliser les touches `h`, `j`, `k` et `l` pour se déplacer respectivement à gauche, en bas, en haut et à droite.

Vim a gardé cette logique, mais il est *(heureusement)* possible d'utiliser les touches directionnelles pour se déplacer.

Dans le fichier `cigale-et-la-fourmi.txt`, placez votre curseur sur la première ligne du texte, entrez en mode **insertion** en pressant `i` et écrivez `La Cigale et la Fourmi` puis appuyez sur `echap` pour revenir en mode **normal**. L'objectif est d'ajouter le titre en début de fichier.

<video style="center" src="videos/mode-i.mp4" controls width="80%" ></video>

Simple ? Une fois de retour en mode **normal**, enregistrez le fichier en pressant `:w` puis `entrée`.

<video style="center" src="videos/save.mp4" controls width="70%" ></video>

Pour quitter vim, `:q` puis `entrée`.

Pour **enregistrer et quitter**, combinez les deux raccourcis précédents : `:wq` puis `entrée`.

Bien que le mode **insertion** permette d'ajouter/supprimer du texte. Nous n'allons pas basculer en **insertion** à chaque fois que nous voulons supprimer une section, ce serait **overkill**.

Auteur

Nous allons alors apprendre quelques raccourcis du mode normal pour nous éviter de constamment changer de mode.

Les deux principales manières de supprimer du texte sont `x` et `d` :

- `x` supprime le caractère sous le curseur.
- `d` supprime une chaine de caractères définie par un paramètre.

Il est également possible de répéter une action en ajoutant un nombre avant la commande. Par exemple, `5x` supprimera les 5 caractères suivants le curseur.

<video style="center" src="videos/x.mp4" controls width="80%" ></video>

Mais avant d'apprendre à supprimer via `d`. Nous allons d'abord **apprendre à nous déplacer de manière efficace**.

En mode **normal**, en pressant `0` ou `^`, le curseur se positionne en début de ligne.

- `gg`, au début du fichier.
- `G`, à la fin du fichier.
- `w`, au début du prochain mot.
- `e`, à la fin du mot actuel/suivant.
- `b`, au début du mot précédent/actuel .
- `:10` ou `10G`, à la ligne 10.

Ces mêmes commandes peuvent servir de paramètre, comme pour l'action `d` *(delete)*.

Essayez alors de supprimer des mots via `dw`, `de` et `db`.

Idem, pour des lignes entières : `d$`, `d0`, ou `dd`

l'instruction `d` est également disponible en tant que commandes *(commençant par* `:` *)*. Pour `d`, la syntaxe est `:[debut],[fin]d`.

Exemple :

- `:5,10d` va tout supprimer entre les lignes 5 et 10.

à retenir qu'il existe certains caractères spéciaux pour définir le début et la fin :

- `.` → La ligne du curseur.
- `$` → la dernière ligne.
- `%` → toutes les lignes.

Si vous voulez annuler une action, utilisez `u` *(undo)*.

Vous savez vous déplacer, supprimer des blocs et refaire une action.

Comme exercice, ouvrez de nouveau `la-cigale-et-la-fourmi.txt`,

- Placez-vous à la 5ème ligne.
- Placez-vous sur ***b****ise* avec une seule action
- Supprimez le *la* en une seule action.
- Supprimez *bise fut* en une seule action. *(répétitions autorisées)*
- Placez-vous à la ligne 10 en une seule action.
- Supprimez les 2 lignes **suivantes** en une seule action.
- Supprimez les 5 lignes **précédentes** en une seule action.

 Rappel: Une répétition est un chiffre X à mettre devant une commande pour qu'elle se répète X fois.

Correction

<video style="center" src="videos/exercice-suppr.mp4" controls width="80%"></video>

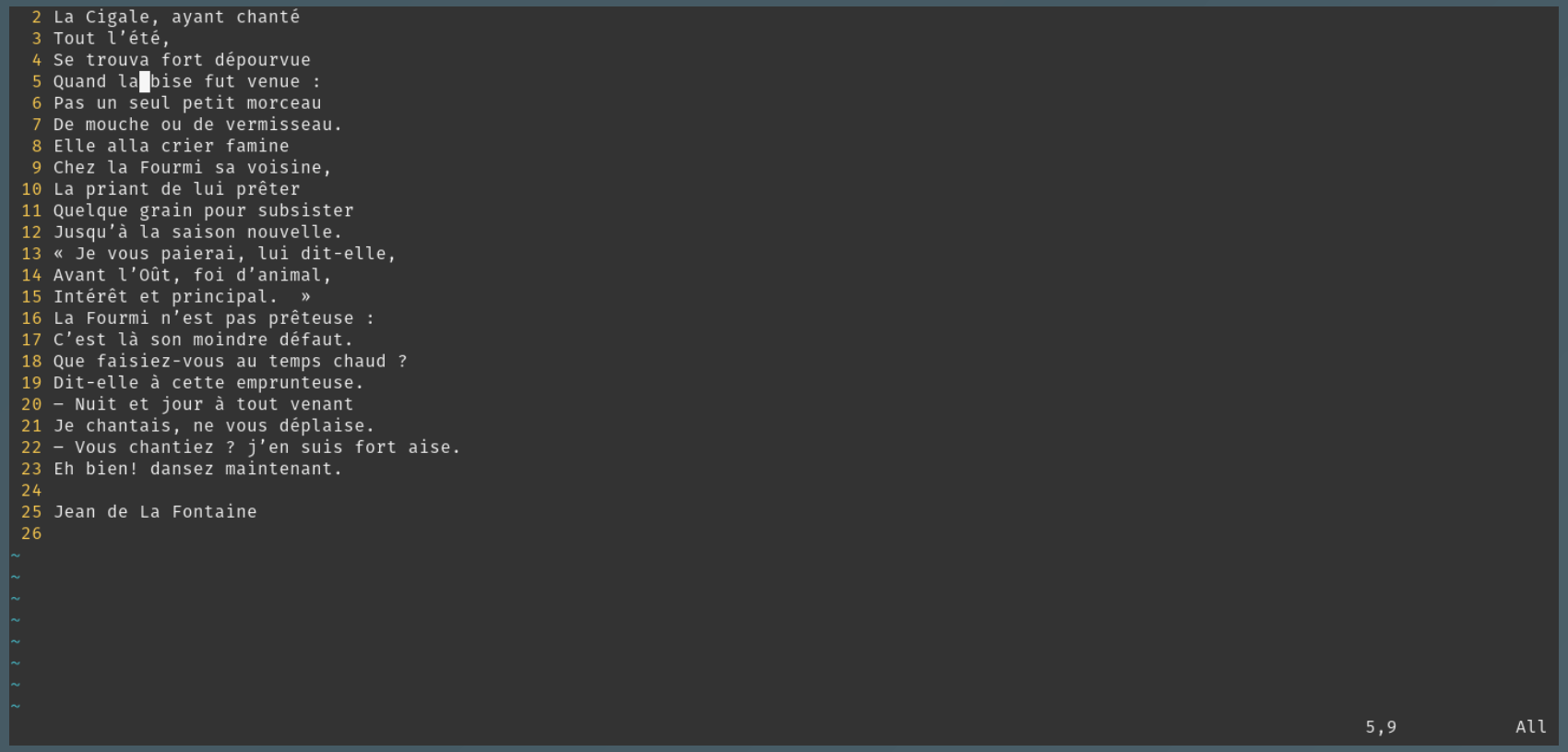
D'autres macros/actions en vrac :

- `r` → remplace le caractère sous le curseur
- `a` → entre en mode insertion après le curseur
- `:.,$-3d` → supprime toutes les lignes entre la ligne du curseur et les 3 dernières lignes du fichier.
- `o` → entre en mode insertion en créant une nouvelle ligne.
- `O` → entre en mode insertion en créant une nouvelle ligne au-dessus.
- `dG` → supprime toutes les lignes du curseur à la fin du fichier.
- `dgg` → supprime toutes les lignes du curseur au début du fichier.

Passer en visuel

Le mode **visuel** est le seul que nous n'avons pas encore abordé. Il permet de sélectionner du texte et y appliquer une macro.

En mode **normal**, appuyez sur `v` pour entrer en mode **visuel**. Vous pouvez alors vous déplacer avec les touches directionnelles pour sélectionner du texte.



Une fois le texte sélectionné, vous pouvez appliquer une macro comme `d` pour couper le texte sélectionné.

👀 Car oui, je vous ai menti ! `d` ne permet pas de **supprimer** mais de **couper**. La différence est que le texte coupé est stocké dans un presse-papier *(clipboard)*.

 C'est l'équivalent total d'un <code>ctrl+x</code>.

En supprimant du texte, il est possible de le coller avec `p` *(paste)* ou `P` pour le coller avant le curseur.

Comme vu dans la précédente animation, il est possible de continuer à utiliser les raccourcis de déplacement en mode **visuel**.

Vous pouvez aussi directement sélectionner la ligne entière avec `V`.

Pour copier une sélection, utilisez `y` *(yank)*.

<video style="center" src="videos/copy-paste.mp4" controls width="70%"></video>

Le *clipboard* de Vim est utilisable entre différents fichiers. C'est-à-dire que vous pouvez copier du texte dans un fichier, puis le coller dans un autre fichier.

<video style="center" src="videos/copy-entre-fichiers.mp4" controls width="80%"></video>

Faire une recherche dans un fichier

Vim intègre une manière simple de faire des recherches : `/` et `?` dont la seule différence est la direction de la recherche. (vers le bas ou vers le haut)

Ces actions prennent en paramètre une **regex** *(expression régulière)*.

À retenir :

- `/` → recherche vers le bas
- `?` → recherche vers le haut
- `n` → recherche le prochain résultat
- `N` → recherche le résultat précédent

Vous pouvez surligner les résultats avec la commande <code>:set hlsearch</code>

Faire une recherche dans un fichier

<video style="center" src="videos/recherche.mp4" controls width="80%"></video>

Vim + sed = ❤️

Sed est un outil en ligne de commande permettant de modifier du texte. Il est très utilisé dans les scripts de déploiement pour modifier des fichiers de configuration. Celui-ci est intégré nativement à vim avec la commande `s`.

En bash : `sed -i 's/old/new/g' file.txt`
Sur Vim : `:[debut],[fin]s/old/new/g`

Comme pour `:d`, `:s` est une commande qui prend en paramètre un début et une fin. Le début et la fin peuvent être des lignes ou des caractères spéciaux comme `.` ou `%`.

Simple remplacement de texte : *Cigale* par *Bestiole* avec `:1,2s/Cigale/Bestiole/`.

<video style="center" src="videos/sed-exemple.mp4" controls width="90%"></video>

Commenter plusieurs lignes via sed

- `v` → entre en mode visuel.
- Sélection des lignes via les flèches.
- `'<,'>s/^/#/` → ajoute un `#` au début de chaque ligne sélectionnée.

<video style="center" src="videos/comment-sed.mp4" controls width="60%" ></video>

 Vim va préremplir les balises du mode visuel <code>'<,'></code>.

D'autres raccourcis en mode visuel:

- (**mode normal**) `gv` → réapplique la dernière sélection visuelle.
- (**mode visuel**) `o` → se place au début/fin de la sélection.
- (**mode visuel**) `u` et `U` → Lowercase/uppercase.
- (**mode visuel**) `~` → Inverse la case.
- (**mode visuel**) `>` et `<` → Ajouter/supprime l'indentation.

Gestion des Buffers

Qu'est ce qu'un Buffer ?

Un **buffer** est un fichier chargé en mémoire. Un fichier ouvert dans Vim est alors un buffer *(qu'il soit affiché à l'écran... ou non)*.

Lorsque nous ouvrons un fichier, celui-ci est chargé dans un buffer. Si nous ouvrons un autre fichier, celui-ci sera chargé dans un second buffer. Nous pouvons alors naviguer entre les fichiers ouverts assez facilement.

Voici quelques commandes utiles pour gérer les buffers :

- `:ls` → liste les buffers ouverts.
- `:b{i}` → ouvre le buffer {i}.
- `:bnext` ou `:bn` → ouvre le buffer suivant.
- `:bprevious` ou `:bp` → ouvre le buffer précédent.
- `:bdelete` ou `:bd` → ferme le buffer courant.

Naviguer entre les buffers

<video style="center" src="videos/buffers.mp4" controls width="80%" ></video>

Maintenant que nous savons *manipuler les buffers*, nous pouvons les utiliser pour rentre leurs usages plus efficaces avec **les onglets** ou le **multifenêtrage**.

Onglets Vim

Ce n'est pas très utilisé mais Vim permet de gérer des onglets afin de travailler sur plusieurs fichiers en même temps. La syntaxe pour ouvrir directement de multiples fichiers en une seule commande est la suivante : `vim -p file1 file2 file3`.

Voici les principales choses à retenir:

- `:tabe` → ouvre un nouveau fichier dans un nouvel onglet.
- `gt` → passe à l'onglet suivant.
- `gT` → passe à l'onglet précédent.
- `{i}gt` → passe à l'onglet {i}.
- `:qa` → ferme tous les onglets.

Démonstration

<video style="center" src="videos/tab-showcase.mp4" controls width="80%" ></video>

Multifenêtrage

Maintenant, on s'attaque à l'affichage de plusieurs buffers en même temps avec le multifenêtrage.

- `:split fichier` ou `:sp` - Ajouter une fenetre horizontale en ouvrant 'fichier'
- `:vsplit fichier` ou `:vsp` - Ajouter une fenetre verticale en ouvrant 'fichier'
- `ctrl-w fleche-haut` - se déplacer vers le haut
- `ctrl-w ctrl-w` - se déplacer vers la fenêtre suivante
- `:resize 10` - redimensionner la fenêtre courante à 10 lignes
- `:vertical resize 10` - redimensionner la fenêtre courante à 10 colonnes
- `:sview` - ouvrir en lecture seule
- `:hide` - fermer la fenêtre courante
- `:only` - fermer toutes les fenêtres sauf la fenêtre courante

Démonstration multifenêtrage

<video style="center" src="videos/multifenetres.mp4" controls width="80%" ></video>

Registres

Lorsque nous manipulons un fichier volumineux, nous allons régulièrement faire des `d` *(couper)* et des `p` *(coller)*.

 Certaines suppressions avec <code>d</code> n'auront pour objectif que de supprimer un bloc de texte, sans volonté de l'enregistrer dans notre presse-papier.
Pourtant Vim va l'enregistrer dans le presse papier et nous allons perdre le contenu qui aurait dû être collé par <code>p</code> (ou <code>P</code>) avant la suppression.

À ce problème, nous avons la solution d'utiliser les **registres**.

Les registres

Lorsque vous supprimez un bloc, Vim va l'ajouter à un registre qui peut retenir jusqu'à **9** blocs automatiquement.

D'autres éléments sont également déjà présents dans le registre en *readonly*:

- `"%` correspond au nom du fichier en cours d'édition
- `":` est la dernière commande exécutée
- `".` affiche la dernière chaine de texte ajoutée
- `"/` correspond à la dernière recherche (sed inclus)
- `"*` correspond au clipboard clic-molette
- `"+` correspond au clipboard système
- `"#` au dernier fichier ouvert (sur la même session)

<p style="font-size:17px"> Vous pouvez voir le contenu de votre

Sauvegarder un élément dans le registre

À présent, vous pouvez également ajouter un élément au registre qui ne sera pas supprimé. Si les clés 0-9 et `%:/*+#` sont en lecture-seule, il vous reste néanmoins toutes les clés alphabétiques.

Chaque commande concernant le registre commence par `"` *(doubles-quotes)*. Pour coller la clé 9, la commande est : `"ap` *(`p` pour coller)*

Pour enregistrer une clé dans le registre: `"ayy` *(`yy` pour copier la ligne)*

Évidemment : `p` et `yy` ne sont pas les seules manières de coller/copier le contenu d'une clé. Les commandes valables dans les précédents cas d'usages sont toujours valides et utilisables dans ce contexte.

à suivre 

n'hésitez pas à me contacter pour toute question, remarque ou suggestion !