TP4 – Rédaction informatique et mathématique – L1 · S1

git pull et push – commandes et styles en LaTeX

Novembre 2024

Objectif

— synchroniser des dépôts git
— gérer des confits de fusion de versions
— définir et redéfinir des commandes LaTeX, simples et complexes
— contrôler la forme en LaTeX (style)

Exercice 1 *Mise en place*

1. Créez un fichier `test.tex` dans un nouveau répertoire nommé `dirA/`, et éditez le pour qu'il contienne un squelette fonctionnel de document LaTeX.
2. Indépendamment, créez un projet vierge sur gitlab.
3. Placé dans le répertoire contenant `dirA/`, clonez une première fois sur votre machine le dépôt créé sur gitlab à la question précédente. Renommer le répertoire créé en `repoB/`.
4. Toujours placé dans le répertoire contenant `dirA/` (et désormais également `repoB/`), en passant un argument supplémentaire à la commande `git clone` (*i.e.*, avec une ligne de commande de la forme `git clone <adresse dépôt gitlab> repoC`), clonez une seconde fois le dépôt gitlab dans un répertoire nommé `repoC/`.

Vous devriez ainsi, comme dans la situation ci-contre (qu'on peut observer avec la commande `tree -aFL 2`), avoir 3 répertoires : deux pour les clones du dépôt créé sur gitlab à la question 2 (`repoB/` et `repoC/`) qui sont donc localement gérés par git, et un répertoire non-géré par git (`dirA/`) contenant le fichier `test.tex` demandé à la question 1.

```
.
├── dirA/
│   └── test.tex
├── repoB/
│   └── .git/
└── repoC/
    └── .git/
```

Dans la suite du TP, nous passerons alternativement d'un répertoire à l'autre. Le fichier `test.tex` servira de bac-à-sable pour découvrir les différentes commandes et subtilités de LaTeX introduites dans le TP, tandis que les deux dépôts seront utilisés pour simuler une collaboration *via* git.

dans `repoB/`

5. Placez vous dans le dépôt `repoB/`. Créez un fichier `.gitignore` et écrivez-y les motifs correspondant aux fichiers générés par la compilation LaTeX – *qu'on veut que git ignore* – comme vu lors du TP précédent. Ajoutez le fichier au dépôt (avec `git add`), et faites une version avec un message adapté (avec `git commit`).
6. Copiez le fichier `dirA/test.tex` dans `repoB/` en le nommant `genealogy.tex`, puis éditez le pour y placer un texte, qui donne une liste de noms plus ou moins reliés à votre généalogie, dans le genre de ce document. [1] Votre document doit compiler.
7. Regardez le statut de votre dépôt git. En particulier, vérifiez que le seul fichier listé par git est le fichier `genealogy.tex` créé à la question précédente et qui est encore inconnu de git. En effet, les autres fichiers générés lors de la compilation devraient être ignorés par git, à cause du fichier `.gitignore` créé à la question 5. Si ce n'est pas le cas, corrigez ce fichier jusqu'à ce que tous les fichiers à ignorer soient ignorés, puis créez une nouvelle version prenant en compte ces corrections (et uniquement cela).

1. Le contenu est peut utile, il faut surtout une liste avec au moins cinq éléments, vos (pseudo) prénom et nom répétés sur chaque élément, accompagné à chaque fois d'un prénom et nom d'une personne différente. Adaptez selon vos familles et/ou votre humeur.

dans repoB/

8. Ajoutez le fichier `genealogy.tex` au dépôt git, et créez une version.
9. Publiez vos versions locales sur gitlab (avec `git push`). Vérifiez sur l'interface en ligne de gitlab que les versions sont bien visibles.

dans repoC/

10. Placez vous dans repoC/ et synchronisez ce dépôt-clone avec gitlab (avec `git pull`). Listez les versions connues localement avec `git log` pour vérifier.

Exercice 2 *Définition de commande*

1. Placez-vous dans repoC/ et éditez le fichier `genealogy.tex`. Définissez, dans le préambule, une commande `\myname` qui ne prend aucun argument et affiche votre nom de famille, suivi de votre prénom. Remplacez toutes les occurrences de votre prénom et nom de famille déjà présentes dans le document par un appel à cette commande.

dans repoC/

2. Sans effacer (ni commenter) la définition précédente, redéfinissez la commande `\myname` pour que le prénom précède le nom. Testez.
3. À l'aide du paquet `xspace`, faites en sorte qu'un espace suive votre nom lorsque votre commande est suivi d'un mot, mais pas lorsqu'elle est suivie d'un signe de ponctuation qui ne nécessite pas d'espace, comme la virgule ou le point. Testez.
4. Versionnez puis publiez sur gitlab.

dans repoB/

5. Placez-vous dans repoB/ et synchronisez le dépôt avec gitlab. Listez les versions.

Police de caractère

Une *police de caractère* (en Anglais, *typeface* ou *font family*) est un ensemble de représentations graphiques de signes typographiques (appelés *glyphes*) pouvant être déclinés en différents corps (la taille), graisses (l'épaisseur du trait, *e.g.*, gras) et formes (*e.g.*, italique). Typiquement, l'ensemble des glyphes couvrent l'ensemble des symboles d'un alphabet ou des caractères `utf8`.[2] Une *fonte* (en Anglais, *font*) est une variation d'une police, c'est à dire une donnée (police + corps + graisse + forme).

Exercice 3 *Police* — *ni repoB/, ni repoC/, ni git ici, mais tout dans dirA/test.tex*

On peut en général regrouper les polices en trois familles principales :

(1) avec empattement (*serif*, on parle de police *roman*) – par exemple la police principale de ce document qui est celle utilisée par défaut par LaTeX et la classe `article` ;

(2) sans empattement (*sans-serif*) – par exemple ici : "sans empattement"[3], ou dans les supports de présentation des cours-td ;

(3) à chasse fixe (*monospace*, c'est à dire que tous les caractères ont la même largeur, on parle aussi de police de type machine à écrire ou *teletype*) – par exemple ici (sur 2 lignes pour bien voir que la largeur des caractères est constante) : `teletype` `epytelet`.

Un document LaTeX possède trois polices principales, une de chaque famille, la police *roman* étant utilisée pour le corps du document. Pour changer de police, on peut utiliser `\textrm`, `\textsf` et `\texttt`, qui prennent un argument auquel est appliqué le changement de police.

2. Nous parlons ici de typographie européenne, voire française. Pour des symboles non-latins, comme dans l'alphabet russe ou les sinogrammes chinois, il faudra utiliser une police particulière, ce qui est bien entendu tout à fait possible en LaTeX.

3. Regardez la base du pied de la lettre p, et comparez avec le texte principal du document, vous comprendrez ce qu'est l'empattement.

1. Écrivez le texte suivant dans `test.tex` en respectant les changements de police.

> Je suis en police par défaut. Je suis en police sans empattement. `Je suis en police à chasse fixe.` Je suis en police roman, explicitement.

Une autre façon de changer de police est d'utiliser les commandes `\rmfamily`, `\sffamily` et `\ttfamily`[4], qui changent la police à partir de l'endroit où elles sont appelées et jusqu'au prochain changement de police ou à la fermeture du bloc courant (*e.g.*, fermeture d'accolade ou fin d'environnement). Ces fonctions s'apparentent donc à un "changement de stylo".

2. À l'aide de ces trois commandes, écrivez l'énumération suivante et la phrase qui suit en respectant les changements de polices.[5] Seuls les éléments de liste 2, 4, 6 et 8 devront contenir une des commandes `\sffamily`, `\rmfamily` et `\ttfamily` (une seule commande chacun).

> 1. Je ne change la police par défaut...
> 2. ... qu'à partir de maintenant, pour une police sans empattement...
> 3. ... qui continue un petit moment...
> 4. ... avant d'être finalement changée...
> 5. ... pour la police par défaut...
> 6. ... puis pour `la police à chasse fixe...`
> 7. `... qui continue...`
> 8. `... encore et encore,` mais pas jusqu'au bout.
>
> Après la liste, je retrouve la police par défaut.

Les trois polices principales (*roman*, sans serif, et *teletype*) sont prises dans un ensemble de polices appelé *extended computer modern* (EC). Il est bien entendu possible de choisir d'autres (ensembles de) polices, par exemple l'ensemble {Times pour *roman*, Helvetica pour *sans serif* et Courier pour *teletype*}, via `\usepackage{times}`.

3. Importez le paquet dans votre préambule, compilez et observez les changements.
4. Retirez l'import de `times` que vous avez ajouté à la question précédenteet vérifiez que vous avez bien spécifié l'encodage de sortie *via* `\usepackage[T1]{fontenc}` dans le préambule (après `\usepackage[utf8]{inputenc}`– encodage d'entrée). Compilez (sans `times` donc), et observez le `pdf`. Zoomez très fortement sur les caractères pour vous apercevoir que la qualité n'est pas au rendez-vous.[6]

Pour les documents en français traitant d'informatique et/ou de mathématique, on préférera les polices vectorielles[7] de l'ensemble *CM super* (que l'on obtiendra via l'import du paquet `cm-super`) ou –mieux pour un `pdf`– de l'ensemble *Latin Modern* (que l'on obtiendra via l'import du paquet `lmodern`).

5. Passez votre document à l'ensemble de police *Latin Modern*, en important le paquet adéquat **après** la spécification de l'encodage de sortie. Compilez. Zoomez. Admirez.

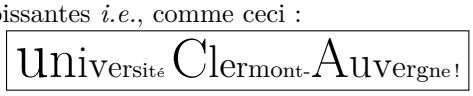
Exercice 4 *Corps* — *ni `repoB/`, ni `repoC/`, ni git ici, mais tout dans `dirA/test.tex`*

Le corps de police, c'est sa taille. LaTeX propose un ensemble de commandes permettant de changer la taille de police (relativement à une taille de police de base pour le document). Ces commandes sont : `\tiny`, `\scriptsize`, `\footnotesize`, `\small`, `\normalsize`, `\large`, `\Large`, `\LARGE`, `\huge` et `\Huge`. À l'instar des commandes `\rmfamily`, `\sffamily`

4. Il existe également des environnements de même noms, qui ont le fonctionnement attendu, mais qui sont peu pratiques et donc peu utilisés.
5. Vous utiliserez la commande `\ldots` partout où il le faut.
6. Au besoin, ouvrez le `pdf` dans un lecteur autre que celui intégré à `texmaker`.
7. qui supportent donc le zoom !

et `\ttfamily` vues à l'exercice précédent, elles fonctionnent sur le mode d'un "changement de stylo" : la taille du texte change à partir de la commande et jusqu'au prochain changement de taille de texte ou à la prochaine fin de bloc.

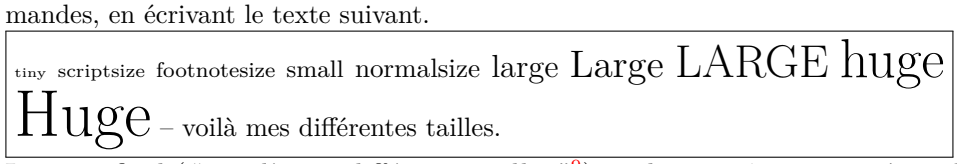
1. Éditez le fichier `test.tex` pour y écrire le nom de l'université avec, pour chaque mot, les lettres en tailles décroissantes *i.e.*, comme ceci :

2. En englobant l'ensemble entre accolade, assurez-vous qu'un texte quelconque que vous ajouterez après n'est pas impacté par le changement de taille (c'est à dire qu'il est en taille normal, sans avoir eu besoin de rajouter `\normalsize`).

Surprenamment, il n'existe pas de commande de base changeant la taille d'un texte **passé en argument** (autrement dit, qui serait à, *e.g.*, `\small`, ce que `\textsf` est à `\sffamily`). Fort heureusement, nous avons vu au dernier cours comment définir nos propres commandes avec arguments.

3. Définissez, pour chaque taille `XXX` [8] donné ci-dessus, une nouvelle commande `\textXXX` qui change la taille du texte passé en argument selon la taille `XXX`. Testez vos commandes, en écrivant le texte suivant.

   Le texte final (*"– voilà mes différentes tailles."* [9]) ne doit pas être impacté par les changements de taille des commandes précédentes (*i.e.*, vous ne devez utiliser ni `\normalsize` ni `\textnormalsize` pour l'écrire à la taille par défaut).

Comme mentionné en début d'exercice, ces différentes tailles sont définis relativement à une taille de police de base pour le document. La plupart des classes LaTeX, dont la classe `article` que vous utilisez, admettent une option indiquant cette taille de base. Cette option est `10pt`, `11pt` ou `12pt`, où `pt` est une unité de longueur : le *point typographique*.

4. Passez votre document en `12pt` ; compilez et observez la différence.

Bien que déconseillé [10], il est possible d'obtenir des tailles non-définies par les commandes précédentes, manuellement *via* la construction :

`\fontsize{<taille>}{<espacement>}\selectfont`

où `<taille>` est la taille de police souhaitée et `<espacement>` indique l'espacement entre les lignes pour la taille de police choisie. [11]

5. Écrivez, en début de document, un texte en plus petit que `tiny` et un texte en plus gros que `Huge`.

Exercice 5 *Graisse*

La graisse c'est ~~la vie~~ l'épaisseur du trait. Principalement, il s'agit de passer du texte en gras, *via* les commandes `\textbf` (qui applique le changement à son argument) ou `\bfseries` (qui "change de stylo"). [12]

dans `repoB/`

1. Dans `repoB/`, passez tous les prénoms en gras, y compris le votre, en préservant, pour les noms de famille, la graisse moyenne (*i.e.*, celle par défaut, qui peut aussi être obtenue avec `\textmd`/`\mdseries`).

8. `XXX` ∈ {`tiny`, `scriptsize`, `footnotesize`, `small`, `normalsize`, `large`, `Large`, `LARGE`, `huge`, `Huge`}.
9. Écrivez `--` pour obtenir le tiret long : '–'.
10. Comme l'ensemble de ce TP d'ailleurs, car le rédacteur devrait théoriquement se concentrer sur le contenu et non la forme...
11. Vous aurez besoin de préciser une unité, *e.g.*, `pt`, `cm`, `mm`, `em`, `ex`, `in`...
12. `bf` signifie ***b**old **f**ace*.

dans repoB/

2. Créez une version prenant en compte les changements de la question précédente, mais **ne la publiez pas** sur gitlab.
3. Dans dirA/test.tex, écrivez la phrase suivante avec ses portions en mode math :
   "Attention, $2+2=2\times 2$ mais $3+3\neq 3\times 3$.
4. Passez la phrase écrite à la question précédente en gras. Que remarquez-vous ?
5. Placez `\mathversion{bold}` à l'intérieur du bloc (délimité par des accolades) qui contient votre phrase. Qu'observez-vous ?
6. Définissez une commande `\strong`, qui passe le texte passé en unique argument (obligatoire) en gras, y compris les parties en mode math. Testez votre commande sur la phrase précédente.

Exercice 6 *Forme*

La forme (en Anglais, *shape*), c'est une variation de la police. Il est ainsi possible de passer du texte en italique (*via* `\textit`/`\itshape`), en incliné (*slanted*, *via* `\textsl`/`\slshape`), en droit (forme par défaut, accessible également *via* `\textup`/`\upshape`), ou en petites capitales (*small caps*, *via* `\textsc`/`\scshape`).

1. Recopiez le texte suivant dans dirA/test.tex.
   L'*italique* est différent du *slanted*. Le *slanted* est différent du texte droit. Le texte droit est très clairement différent du TEXTE EN PETITES CAPITALES.

dans repoC/

2. Dans repoC/, changez tous les nom de famille du document `genealogy.tex` pour qu'ils soient affichés en petites capitales (avec tout de même une majuscule (ou grande capitale) en début de nom).
3. Créez une version traçant ces changements, mais **ne la publiez pas** sur gitlab.
4. S'il est possible de combiner une forme, une graisse et une taille, toutes les combinaisons ne sont en général malheureusement pas disponibles (selon la police choisie). Parmi les combinaisons suivantes, que vous devez essayer, lesquelles sont possibles ?
   a) gras et italique ;
   b) gras et slanted ;
   c) gras et en petites capitales.

La commande `\emph`, abrégeant *emphasize* qui signifie "mettre en valeur", est le plus souvent définie pour passer le texte à mettre en valeur en italique. Cependant, si ce texte se trouve à l'intérieur de texte déjà en italique, alors il est mis en valeur en étant repassé en texte droit.

5. Observez ce comportement.

Il faut (presque) toujours préférer `\emph` à `\textit` (et `\em` à `\itshape`), d'une part pour que le comportement décrit ci-dessus se produise, et d'autre part parce que la paire `\emph`/`\em` est orientée contenu tandis que `\textit`/`\itshape` est orientée mis-en-forme.

Exercice 7 *Synchronisation*

Les modifications apportées au projet lors de l'exercice 5 dans repoB/ et lors de l'exercice 6 dans repoC/ n'ont pas été publiées sur gitlab et ont encore moins été partagées entre les deux dépôts. Ceux-ci ont donc évolué indépendamment et il est temps de les synchroniser.

1. Vérifiez que les deux dépôts (repoB/ et repoC/) ont un statut propre (aux yeux de git). Au besoin, regardez les changements non pris en compte *via* `git diff`, et créez une version pour les inclure.

dans repoB/

2. Publiez les versions du dépôt repoB/ sur gitlab.

dans repoC/

3. Tentez de publier les versions du dépôt repoC/. Que se passe-t-il ? Pourquoi ?

4. Récupérez les versions depuis gitlab dans `repoC/`. Avez-vous des conflits à résoudre (dans `repoC/`) ? Si oui : résolvez-les à la main (en éditant le fichier, dans lequel les zones conflictuelles ont été marquées par git) et concluez la fusion (avec `git add <fichier>` suivi de `git commit` sans message). Si non : refaites le TP...

dans `repoC/`

5. Publiez les versions de `repoC/` sur gitlab.

dans `repoB/`

6. Récupérez les versions depuis gitlab dans `repoB/`.
7. Dans chacun des deux dépôts, listez les versions sous forme de graphe, *via* la commande `git log --graph --oneline` et comprenez ce qu'il s'est passé.

Pour éviter des éventuelles futures fusions pénibles comme celle rencontrée à la question 4, nous allons factoriser un peu le code LaTeX.

8. Dans le préambule de `repoC/genealogy.tex`, créez une commande `\mystyle` qui prend 2 arguments, dont un (le premier) optionnel avec valeur par défaut vide. Le premier argument sera vu comme un style (une suite de commandes qui "changent de stylo"), et le second argument comme un texte à styliser. La commande doit donc appliquer le style donné en premier argument au texte donné en second argument. Ainsi, le code :

   ```
   Ça, \mystyle[\bfseries\slshape]{c'est} du \mystyle[\scshape\large]{style} !
   ```

   dans `repoC/`

   donnera :

   Ça, ***c'est*** le STYLE !
9. Définissez une commande `\name`, qui prend deux arguments obligatoires (un nom de famille et un prénom), et applique les styles choisis précédemment à chacun de ces deux arguments, en appelant la commande `\mystyle` définie à la question précédente.
10. Modifiez tous les occurrences de nom et prénom présentes dans le document afin de faire appel à la commande `\mystyle`.

Ainsi, les styles que l'on voudra appliquer à tous les noms et tous les prénoms, et même l'ordre dans lequel on voudra donner ces deux informations, pourront être définis une fois pour toute dans le préambule (dans la définition de `\mystyle`). Ceci évitera d'avoir des changements dans tout le fichier et donc des fusions difficiles... Cela va en outre dans le bon sens : la partie contenu du document ne se concentre que sur le contenu (pas sur la mise-en-forme).

dans `repoC/`

11. Une fois le résultat désiré obtenu, faites une version et publiez la sur gitlab.

dans `repoB/`

12. Récupérez les versions depuis gitlab dans `repoB/`.

Exercice 8 *Coloration*

Il est bien sûr possible de changer la couleur du texte. Pour cela on peut importer le paquet `xcolor`. Il existe deux commandes de base pour changer la couleur d'un texte, à savoir, `\textcolor` et `\color`. Les deux prennent en premier argument le nom de la couleur [13] à utiliser. La première commande prend comme second argument le texte à colorier, tandis que la seconde commande ne prend pas de second argument mais change la couleur du texte jusqu'au prochain changement de couleur ou la prochaine fin de bloc (elle fait "un changement de stylo").

1. Importez le paquet `xcolor`, puis écrivez du texte en bleu, en rouge, en vert, en gris, en violet, *etc*...
2. Dans `repoB/`, changez le style des noms de l'ensemble du document `genealogy.tex`, pour que les prénoms soient en gris, et les noms de famille en bleu. Faites une version, et publiez la sur gitlab. Possiblement (probablement), vous devriez avant cela récupérer les versions publiées depuis `repoC/` à l'exercice précédent.

dans `repoB/`

13. Les noms de couleurs en anglais et en minuscule fonctionnent pour la plupart des couleurs.

dans `repoB/`

3. Modifiez à nouveau le document `genealogy.tex` afin de définir une couleur nommée `firstnamecolor` pour les prénoms et une autre, nommée `lastnamecolor`, pour les noms de famille. Ces définitions peuvent utiliser la commande `\colorlet`. Changez le style des noms et prénoms pour qu'ils utilisent la couleur appropriée. Testez. Versionnez & publiez vos travaux sur gitlab.
4. En prenant garde à vos yeux, utilisez, dans `test.tex`, la commande `\pagecolor` définie par le paquet `xcolor`, qui prend en unique argument une couleur. Que fait cette commande ?
5. Importez le paquet `hyperref`, avec les options `colorlinks=true` [14] et `urlcolor=blue`. Créez un lien hypertexte vers la page web de votre choix, compilez et vérifiez que le rendu est tel qu'attendu.
6. Le paquet `soulutf8` (une amélioration gérant l'encodage utf8 du paquet `soul`) permet (entre autre) de surligner du texte, via la commande `\hl`. Surlignez du texte (dans `test.tex`), avec la couleur par défaut. Quelle est-elle ?
7. À l'aide de `\sethlcolor`, surlignez du texte en vert.
8. Définissez une commande `\surligne` qui prend un premier argument optionnel, dont la valeur par défaut est `pink`, et un second argument obligatoire, et qui surligne le texte donné en second argument dans la couleur précisée en premier argument. Testez avec et sans argument optionnel.
9. Vérifiez que l'utilisation de la commande `\hl` après l'utilisation de votre commande, conserve sa couleur par défaut (qui n'est ni verte ni rose).
10. Définissez une commande `\bgfg` qui prend trois arguments obligatoires, à savoir, dans cet ordre, une couleur d'arrière plan (***b**ack**g**round* pour le surlignage), une couleur de premier plan (***f**ore**g**round* pour la couleur de texte), et un texte à colorier de la sorte. Ainsi, `\bgfg{pink}{red}{blabla}` devra donner cette horreur : blabla.

Je vous épargne le soulignage qui n'est de toute façon pas une très bonne idée.
Allez voir du côté de `soulutf8` (ou, alternativement, `ulem`) si vous y tenez.

---

14. `colorlinks` tout court suffit.