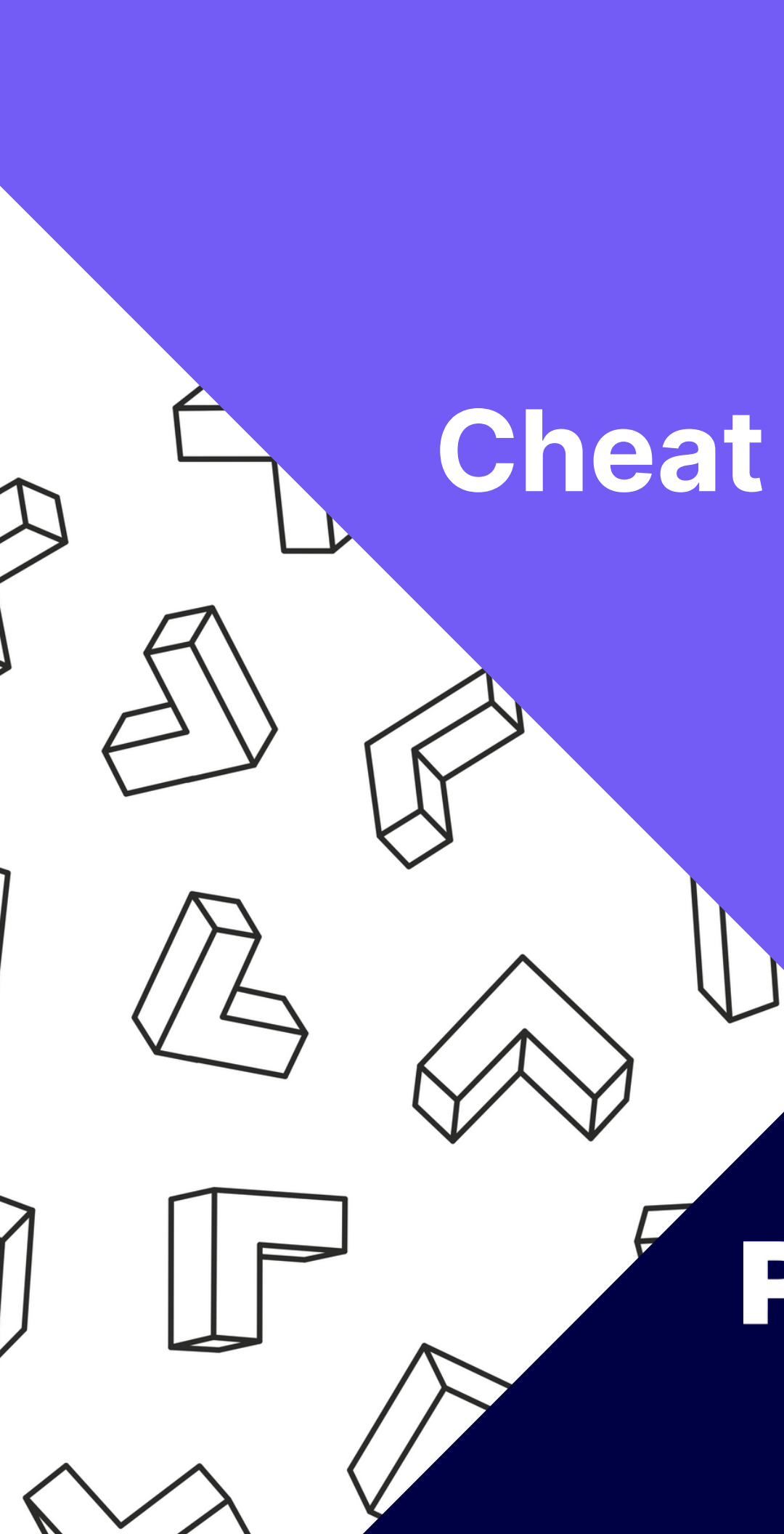
GIT : Cheat Sheet



PETAL

Table des matières

01 COMMANDES

Communes

git init	Créer un repository sur votre environnement (local)
git status	Voir ce que vous êtes en train de faire (fichiers modifiés)
git add	Ajouter des fichiers au stage
git commit	Enregistrer une modification individuelle dans un fichier ou un ensemble de fichiers
git stash	Cacher les modifications indésirables dans un espace de travail

Branches

git checkout	Basculer vers la branche spécifiée, l'option -b créera une nouvelle branche
git merge	Prendre les modifications d'une branche et les appliquer dans une autre

Synchronisation

git clone	Copier un repository (remote) sur votre environnement (local)
git fetch	Ajouter les modifications du référentiel distant à votre branche de travail locale
git push	Envoyer vos commits vers un repo distant
git pull	Récupérer les modifications et les merger

01 COMMANDES

Historique

git rebase	Appliquer une série de modifications d'une branche à une base différente
git cherry-pick	Choisir un sous-ensemble de modifications à partir d'une série de modifications et enregistrez-les comme une nouvelle série de modifications sur une base de code différente

Annulation

git revert	Annuler des commits
git restore	Annuler les modifications
git reset	Réinitialiser HEAD (branche active) à l'état spécifié

Aperçu

git diff	Un diff est la différence de modifications entre deux commits
git log	Voir les commit logs
git show	Affiche un ou plusieurs objets (blobs, trees, tags et commits)

Aide

git help everyday	Git en une vingtaine de commandes pratiques

Voici un lien pratique pour plus d'informations

02 PRODUCTIVITÉ

Aliases

Les alias Git sont des raccourcis vers des commandes Git. En créant des alias pour vos commandes les plus fréquemment utilisées, vous deviendrez plus rapide et plus efficace.

Les alias peuvent être utilisés pour encapsuler une séquence de commandes Git dans une nouvelle fake commande Git.

Lien pratique

Quelques exemples

git s : git status
git cm : git commit -m
git co : git checkout / cob : git checkout -b
git del : git branch -D
git discard-all = !git add . && git reset --hard

Lien pratique

Comment les créer

1- Directement dans le fichier de configuration .gitconfig

```
[alias]
    s = status
    cm = commit -m
    co = checkout
    cob = checkout -b
    del = branch -D
    discard-all = !git add . && git reset --hard
```

2- En utilisant la commande de configuration

```
git config --global alias.co checkout
```

02 PRODUCTIVITÉ

Auto-correction

Il est possible d'activer un système d'auto-correct pour corriger les petites erreurs dans les commandes
`git config --global help.autocorrect 20`

L'auto-correction s'effectuera après 2 secondes.

```
> git cmmit
> WARNING: You called a Git command named 'cmmit', which does not exist.
Continuing in 2 seconds, assuming that you meant 'commit'.
```

Lien pratique

Auto-completion

L'auto-complétion est essentielle si vous utilisez Git CLI et que vous souhaitez sauver du temps. La complétion par tab est une fonctionnalité intéressante, de nombreux shells et différentes configurations sont disponibles, vous n'avez qu'à les appliquer pour en profiter.

Lien pratique

03 BONNES PRATIQUES

Commitez souvent

Les commits peuvent être vu comme des save points. Au minimum, votre travail devrait être commité (et pushed) une fois par jour. Idéalement, vous devriez faire plusieurs checkpoints pour segmenter votre travail et/ou pour vous assurer d'avoir une base à laquelle revenir si jamais les choses venaient à tourner au vinaigre.

Commitez seulement ce qui est pertinent

Un commit ne devrait pas nécessairement contenir tous les fichiers qui ont été changés. Il ne devrait pas non plus nécessairement comprendre toutes les modifications apportées à un seul fichier. Assurez-vous de compartimenter votre travail en blocs logiques afin d'en faciliter la compréhension et la reconstitution.

Écrivez des messages pertinents

Vos commit messages devraient être courts, concis, et indiquer le travail effectué. Les messages génériques ou de seulement quelques mots devraient être évités.

Utilisez des branches

Ne pas push directement dans master vous permettra de garder le codebase "production ready" et de travailler en parallèle.

Gardez vos branches à jour

Garder votre code à jour, peu importe la source de votre branche simplifiera votre travail lorsque le temps viendra de la merger avec sa base. Cela vous évitera également d'accidentellement altérer le travail d'autrui qui entrerait en conflit avec le vôtre.

03 BONNES PRATIQUES

Ne paniquez pas

Tant que vous avez commité votre travail, il ne sera pas perdu. L'annulation, la correction ou la suppression de commits sont toujours des options. Aussi bien dire que le voyage dans le temps est possible!

Connaissez l'outil

N'hésitez pas à tirer profit de ce que Git et la plateforme de votre choix ont à offrir. La documentation est à portée de main et il est facile d'expérimenter sans craindre d'impacter pour toujours et sans billet de retour vos projets.

04 APPRENDRE ET PRATIQUER EN LIGNE

Consultez ces liens pratiques

- https://learngitbranching.js.org/
- https://gitexercises.fracz.com/
- https://www.pluralsight.com/courses/code-school-git-real
- https://www.katacoda.com/courses/git