Jenga2 – Commands

Le module **Commands** contient l'ensemble des **commandes CLI** exposées par Jenga2.
Chaque commande est implémentée dans un fichier Python distinct, suit une interface commune et peut être exécutée via le point d'entrée `jenga2`.

🐚 Sommaire

🧬 Architecture des commandes

Chaque commande est une **classe Python** contenant une méthode statique `Execute(args: List[str]) -> int`.
Cette méthode reçoit la liste des arguments (sans le nom de la commande) et retourne un code de sortie (0 = succès, autre = échec).

Toutes les commandes sont enregistrées dans le dictionnaire `COMMANDS` du module `__init__.py`, avec leurs alias.
Le fichier `jenga2.py` (point d'entrée) utilise le dispatcher `execute_command()` pour appeler la commande appropriée.

```
jenga2.py ──> Commands.__init__.execute_command() ──> Classe de
commande.Execute(args)
```

Avantages :

- Interface uniforme.
- Chaque commande peut avoir son propre parser `argparse`.
- Facile à étendre.

rihen.universe@gmail.com • teuguiasederis@gmail.com
(+237) 693-761-773

📋 Liste complète des commandes

Commande	Alias	Description
`build`	`b`	Compile le workspace ou un projet spécifique
`run`	`r`	Exécute l'exécutable d'un projet (build auto si nécessaire)
`test`	`t`	Lance les tests unitaires (projets `TEST_SUITE`)
`clean`	`c`	Supprime les fichiers générés (objets, binaires)
`rebuild`		Nettoie puis compile
`watch`	`w`	Surveille les fichiers et rebuild automatiquement
`info`	`i`	Affiche les informations du workspace et des toolchains
`gen`		Génère des fichiers projet pour CMake, Makefile, Visual Studio
`workspace`	`init`	Crée un nouveau workspace
`project`	`create`	Crée un nouveau projet (ou un élément dans un projet existant)
`file`	`add`	Ajoute des fichiers, includes, librairies ou defines à un projet
`install`	`i`	Installe les dépendances externes (via `include`)
`keygen`	`k`	Génère une keystore pour la signature Android
`sign`	`s`	Signe un APK Android ou IPA iOS
`docs`	`d`	Génère la documentation Doxygen (Markdown/HTML/PDF)
`package`		Crée des packages distribuables (APK, IPA, MSI, DEB, DMG, ...)
`deploy`		Déploie l'application sur un appareil (adb, ios-deploy, xbapp, ...)
`publish`		Publie un package sur un registre (NuGet, vcpkg, conan, ...)
`profile`		Lance un profilage de performance (perf, Instruments, ...)
`bench`		Exécute des benchmarks (Google Benchmark, ...)
`help`	`h`	Affiche l'aide générale ou celle d'une commande

🚀 Utilisation générale

```
jenga2 <commande> [options]
# ou via alias
jenga2 b --config Release
```

Chaque commande supporte l'option `--help` (ou `-h`) pour afficher sa propre aide :

```
jenga2 build --help
```

🔍 Détail par commande

1. Commandes de build

`jenga2 build`

```
jenga2 build [--config NAME] [--platform NAME] [--target PROJECT] [--no-cache]
[--verbose]
```

- `--config` : configuration de build (`Debug`, `Release`, ...). Défaut : `Debug`.
- `--platform` : plateforme cible (`Windows`, `Linux`, `Android-arm64`, ...). Défaut : hôte.
- `--target` : ne compiler qu'un projet spécifique et ses dépendances.
- `--no-cache` : ignorer le cache et forcer le rechargement complet.
- `--verbose` : affichage détaillé.

Comportement :

1. Tente d'utiliser le **daemon** (si disponible et `--no-daemon` non précisé).
2. Sinon, charge le workspace (via `Cache` ou `Loader`), crée le builder approprié et exécute `Builder.Build()`.

`jenga2 run`

```
jenga2 run [PROJECT] [--config NAME] [--platform NAME] [--args ...] [--no-
build]
```

- Lance l'exécutable du projet (build automatique sauf `--no-build`).
- Les arguments après `--args` sont passés à l'exécutable.

`jenga2 clean`

```
jenga2 clean [--config NAME] [--platform NAME] [--project NAME] [--all]
```

- Supprime **uniquement les fichiers générés** (objets, binaires) sans supprimer les répertoires partagés.
- `--all` : supprime tout le dossier `Build/` et le cache.

`jenga2 rebuild`

```
jenga2 rebuild [options]
```

- Exécute `clean` puis `build` avec les mêmes options.

`jenga2 watch`

```
jenga2 watch [--config NAME] [--platform NAME] [--polling] [--no-daemon]
```

- Surveille les fichiers du workspace.
- Si un changement est détecté, lance automatiquement un build.
- Utilise le **daemon** (sauf `--no-daemon`) pour une réactivité maximale.

---

2. Commandes de test

`jenga2 test`

```
jenga2 test [--config NAME] [--platform NAME] [--project NAME] [--no-build]
```

- Recherche tous les projets de type `TEST_SUITE` (ou `isTest = True`).
- Les compile et les exécute.
- Intègre le framework **Unitest** (voir `Unitest/README.md`).

---

3. Commandes de projet

`jenga2 workspace` (alias `init`)

```
jenga2 workspace [NAME] [--path DIR] [--configs LIST] [--oses LIST] [--
interactive]
```

- Crée un nouveau workspace avec un fichier `.jenga` nommé `NAME.jenga`.
- Mode interactif si `--interactive` ou si aucun argument.
- Définit les configurations, OS cibles et architectures par défaut.

`jenga2 project` (alias `create`)

```
jenga2 project [NAME] [--kind KIND] [--lang LANG] [--location DIR] [--
interactive]
jenga2 project --element TYPE --name NAME --project PROJECT [--template TMPL]
```

- **Création de projet** : ajoute un nouveau projet au workspace courant.
- **Création d'élément** : génère un fichier avec template (class, struct, enum, union, function, custom).

Exemples :

```
jenga2 project Game --kind console
jenga2 project --element class --name Player --project Game
```

`jenga2 file` **(alias** `add`**)**

```
jenga2 file [PROJECT] [--src FILES] [--inc DIRS] [--link LIBS] [--def DEFINES]
[--type TYPE]
```

- Ajoute des ressources à un projet existant.
- Modifie **directement** le fichier `.jenga` en insérant les appels de fonction dans le bloc du projet.
- Supporte les types `source`, `header`, `resource`.

---

4. Commandes de documentation

`jenga2 docs`

```
jenga2 docs extract [--project NAME] [--output DIR] [--format
markdown|html|pdf|all] [--include-private]
jenga2 docs stats [--project NAME]
jenga2 docs list
jenga2 docs clean [--project NAME] [--output DIR]
```

- Analyse les sources C++ (`.h`, `.cpp`, `.hpp`, `.inl`, ...), extrait les signatures et les commentaires Doxygen.
- Génère une documentation Markdown structurée (index, fichiers, namespaces, types, recherche, statistiques).
- Les autres formats (HTML, PDF) sont à implémenter.

Framework d'extraction complet : supporte les `@brief`, `@param`, `@return`, `@note`, `@warning`, `@see`, `@since`, `@deprecated`, `@code`, les modules C++20, etc.

---

5. Commandes de packaging et déploiement

`jenga2 package`

```
jenga2 package --platform android --type apk [--config Release] [--project
NAME] [--output DIR]
jenga2 package --platform ios   --type ipa
jenga2 package --platform windows --type msi|exe|zip
jenga2 package --platform linux   --type deb|rpm|appimage|snap
jenga2 package --platform macos   --type pkg|dmg
jenga2 package --platform web     --type zip
```

- Crée des **installateurs** ou **archives** prêts à la distribution.
- Utilise les outils natifs : WiX / Inno Setup (Windows), dpkg-deb (Linux), pkgbuild/create-dmg (macOS), apksigner/bundletool (Android), etc.
- Nécessite l'installation préalable des outils (message d'erreur explicite si absent).

`jenga2 deploy`

```
jenga2 deploy --platform android [--target DEVICE_ID]
jenga2 deploy --platform ios    [--id UDID]
jenga2 deploy --platform xbox   [--target IP]
jenga2 deploy --platform macos  [--target /Applications]
```

- Déploie l'application sur un appareil physique ou un émulateur.
- **Android** : via `adb install -r`.
- **iOS** : via `ios-deploy`.
- **Xbox** : via `xbapp deploy`.
- **macOS** : copie dans `/Applications` (si `--target /Applications`).

`jenga2 publish`

```
jenga2 publish --registry nuget --package MonApp.nupkg --api-key XXXX
jenga2 publish --registry npm   --package . --api-key YYYY
```

- Publie un package sur un registre.
- Support actuel : **NuGet** (`dotnet nuget push`).
- Extensible pour vcpkg, Conan, npm, PyPI.

---

6. Commandes avancées

`jenga2 keygen`

```
jenga2 keygen [--alias mykey] [--validity 10000] [--output keystore.jks] [--
interactive]
```

- Génère une **keystore Android** via `keytool`.
- Mode interactif pour saisir mot de passe, DN, etc.

`jenga2 sign`

```
jenga2 sign --apk app.apk --keystore keystore.jks --alias mykey
jenga2 sign --ipa app.ipa --identity "Apple Development: ..."
```

- Signe un APK Android (via `apksigner`) ou un IPA iOS (via `codesign`).

`jenga2 profile`

```
jenga2 profile --platform linux --tool perf --duration 30
jenga2 profile --platform macos --tool instruments
```

- Lance un outil de profilage (perf, Instruments, ...).
- **Stade actuel** : base pour `perf` sur Linux, à enrichir.

`jenga2 bench`

```
jenga2 bench [--project MonBench] [--iterations 10] [--output results.json]
```

- Exécute un projet de benchmark (Google Benchmark, etc.).
- Suppose que l'exécutable accepte `--benchmark_out=...`.

🏷️ Gestion des alias

Le fichier `__init__.py` maintient deux dictionnaires :

```
COMMANDS = {
    'build': BuildCommand,
    'run': RunCommand,
    ...
}
```

```
ALIASES = {
    'b': 'build',
    'r': 'run',
    't': 'test',
    'c': 'clean',
    'w': 'watch',
    'i': 'info',
    'init': 'workspace',
    'create': 'project',
    'add': 'file',
    'k': 'keygen',
    's': 'sign',
    'd': 'docs',
    'h': 'help',
}
```

L'utilisateur peut donc taper `jenga2 b` au lieu de `jenga2 build`.

Intégration avec le daemon

La plupart des commandes **peuvent** utiliser le daemon pour une exécution quasi-instantanée.

- **Si le daemon tourne** et que l'option `--no-daemon` n'est pas spécifiée, la commande envoie une requête RPC au daemon et reçoit le résultat.
- **Sinon**, la commande exécute le workflow traditionnel (chargement du workspace, build, etc.).

Implémentation : chaque commande teste `DaemonClient.IsAvailable()` et appelle `client.SendCommand(...)` en cas de succès.

Créer une nouvelle commande

1. **Créer un fichier** `MaCommande.py` dans `Commands/`.
2. **Définir une classe** `MaCommandeCommand` avec une méthode statique `Execute(args: List[str]) -> int`.
3. **Ajouter l'import et l'enregistrement** dans `Commands/__init__.py` :

```
from .MaCommande import MaCommandeCommand

COMMANDS['macommande'] = MaCommandeCommand
ALIASES['mc'] = 'macommande'
```

4. **Optionnel** : ajouter l'aide dans `HelpCommand._ShowGlobalHelp()` et dans la docstring de la classe.

Exemple minimal :

```python
import argparse

class MaCommandeCommand:
    @staticmethod
    def Execute(args):
        parser = argparse.ArgumentParser(prog="jenga macommande")
        parser.add_argument("--option", help="Une option")
        parsed = parser.parse_args(args)
        print(f"Option = {parsed.option}")
        return 0
```

☑ Bonnes pratiques

- **Toujours** fournir une aide via `argparse`.
- **Retourner un code de sortie** (0 = succès, ≠0 = échec).
- **Utiliser les utilitaires** `Colored`, `Display`, `Reporter` pour un affichage cohérent.
- **Privilégier le daemon** si disponible (gain de temps).
- **Documenter** les paramètres, le comportement et les exemples dans la docstring.
- **Ajouter des alias courts** si la commande est fréquente.

🔗 Liens connexes

- Documentation Core
- Builders
- Utilitaires Jenga2
- Système de tests Unitest
- Guide utilisateur

Ce document est maintenu par l'équipe Jenga. Toute suggestion d'amélioration est la bienvenue.