🐳 Docker Cheatsheet – cp, Bind Mounts & Volumes

◇ 1. `docker cp` – Datei vom Host in den Container kopieren

☑ Ziel:

Dateien vom Host in einen laufenden Container verschieben und dort weiterverwenden.

📋 Vorbereitung:

```
echo "Hallo, das ist eine Testdatei für Docker und M347!" > meine_datei.txt
docker run -d --name mein_container ubuntu sleep 6000
```

📦 Datei kopieren:

```
docker cp meine_datei.txt mein_container:/root/
```

🔍 Überprüfung im Container:

```
docker exec -it mein_container bash
cd /root/
ls
cat meine_datei.txt
exit
```

📄 Erfolgsnachweis:

- Datei ist im Container vorhanden (`/root/meine_datei.txt`)
- Inhalt wird korrekt angezeigt

🧹 Bonus – Aufräumen:

```
docker stop mein_container
docker rm mein_container
```

◇ 2. Bind Mounts – Ordner auf dem Host im Container verfügbar machen

☑ Ziel:

Ordner und Dateien vom Host mit dem Container synchron halten.

📋 Vorbereitung:

```
mkdir mein_mount
cd mein_mount
echo "Hallo, dies ist eine Datei für Bind Mounts!" > testdatei.txt
```

🚀 Container starten mit Bind Mount:

```
docker run -d --name mount_container -v ${pwd}:/mnt/daten ubuntu sleep 6000
```

🔍 Überprüfung im Container:

```
docker exec -it mount_container bash
cd /mnt/daten
ls
cat testdatei.txt
```

🖊️ Änderung auf dem Host:

```
echo "Diese Zeile wurde nachträglich hinzugefügt!" >> mein_mount/testdatei.txt
```

🔁 Check im Container:

```
cat /mnt/daten/testdatei.txt
```

🧾 Erfolgsnachweis:

- Datei sichtbar & editierbar im Container
- Änderungen vom Host sofort im Container sichtbar

🩹 Bonus – Datei im Container erstellen und auf dem Host prüfen:

```
touch /mnt/daten/container_datei.txt
exit
```

```
ls mein_mount
```

◇ 3. Bind Mounts mit `-v` Syntax (explizit & Rechte testen)

📋 Vorbereitung:

```
mkdir \mein_mount
echo "Hallo, dies ist eine Datei für Bind Mounts mit -v!" >
\mein_mount\testdatei.txt
```

🚀 Container starten:

```
cd \mein_mount
docker run -d --name mount_container -v ${pwd}:/mnt/daten ubuntu sleep 6000
```

🔍 Container prüfen:

```
docker exec -it mount_container bash
cd /mnt/daten
ls
cat testdatei.txt
```

📝 Datei im Container bearbeiten:

```
echo "Diese Zeile wurde im Container hinzugefügt!" >> /mnt/daten/testdatei.txt
```

📤 Host prüfen:

Öffne die Datei `\mein_mount\testdatei.txt` mit z. B. `notepad` oder `cat`.

📄 Erfolgsnachweis:

- Datei sichtbar im Container
- Änderungen beidseitig synchronisiert

🔐 Bonus – Read-Only Bind Mount:

```
docker run -d --name mount_readonly -v ${pwd}:/mnt/daten:ro ubuntu sleep 60
docker exec -it mount_readonly bash -c "echo 'Test' > /mnt/daten/neue_datei.txt"
```

🟤 **Erwartung**: Fehlermeldung wegen `read-only` → gute Grundlage zur Diskussion

---

◇ 4. Volumes (Kurzfassung für Kontext im Vergleich)

> Volumes sind persistente Datencontainer, die unabhängig vom Lifecycle des Containers existieren.

Anlegen & Verwenden:

```
docker volume create mein_volume
docker run -d --name vol_container -v mein_volume:/app/data ubuntu sleep 6000
```

Check:

```
docker exec -it vol_container bash
cd /app/data
touch test_vol.txt
ls
```

🟤 **Vorteile von Volumes**:

- Persistenz auch bei Container-Neustart
- Besser für komplexe Datenspeicherung
- Einfaches Backup möglich

🔄 5. Vergleich & Anwendungsfälle

Methode	Einsatzbereich	Persistenz	Synchronisierung	Zugriff
`docker cp`	Einzelne Dateien kopieren	Nein	Keine	Einmalig
Bind Mounts	Echtzeit-Zugriff auf lokale Dateien	Ja	Sofortig	Beidseitig
Volumes	Langfristige Datenspeicherung	Ja	Automatisch	Container-Only

---

🧠 6. Reflexion & Abschluss

📌 **Mögliche Diskussionspunkte:**

- Wann setze ich welche Methode ein?
- Welche Variante ist am sichersten / flexibelsten?
- Wie beeinflussen Dateiberechtigungen die Arbeit?