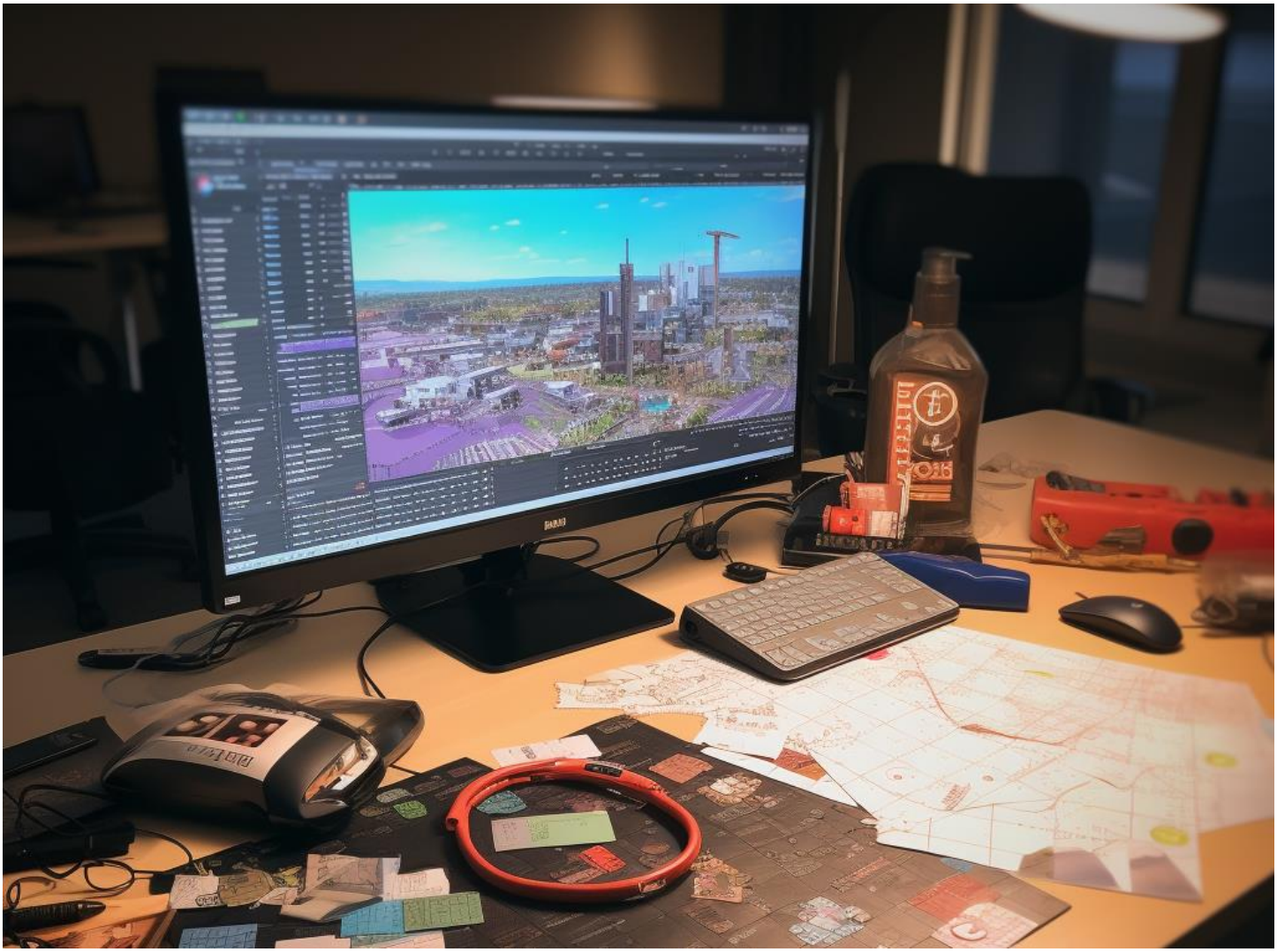
Useful Git Commands
That Will Make Your Life Easier

Git is an invaluable tool for developers, allowing version control and collaboration. While many are familiar with basic commands like commit and push, there are numerous lesser-known commands that can simplify your workflow. Let's dive into them.

1. git stash:



Imagine you're coding a new feature and suddenly need to address a critical bug in another branch. Instead of committing unfinished work, you can use git stash to temporarily save your changes. Once the bug is fixed, use git stash apply to retrieve your saved work. Linus Torvalds, the genius behind Git, once mentioned, "Most people are afraid of 'git stash'. They're wrong."

Examples:

```
$ git stash save "Saving work before switching branches"
```

```
$ git stash apply
```

2. git log --oneline:

When managing a project with numerous commits, git log can appear chaotic. For a concise view, git log --oneline lists each commit as a single line, showing the commit ID and the commit message side-by-side.

Examples:

```
$ git log --oneline
```

Output:

```
e5f6cbe Fix typo in documentation
d2a569b Add new feature X
b3f5c8a Initial commit
```

3. git blame [filename]:

Suppose you're viewing a file and find an odd piece of code. Who wrote it? When? Using git blame filename, you'll see line-by-line annotations indicating the author and commit.

Examples:

```
$ git blame main.js
```

Output:

```
e5f6cbe (Alice 2023-07-20) function newFeature() {...}
d2a569b (Bob   2023-07-15) function oldFeature() {...}
```

4. git reflog:

If you've mistakenly deleted a commit, all might seem lost. However, git reflog tracks every change made to branch tips. By identifying the commit hash, you can recover your work using git checkout [commit_hash].

Examples:

```
$ git reflog
```

Output:

```
e5f6cbe HEAD@{0}: commit: Fix typo in documentation
d2a569b HEAD@{1}: checkout: moving from feature-branch to master
```

5. git cherry-pick [commit_hash]:

Maybe you've made a commit in a branch (let's say feature-branch) that you now want in master. Instead of merging the entire branch, you can pick just that commit using git cherry-pick commit_hash.

Examples:

```
# Assuming you're on master and want to get a commit from feature-branch
$ git cherry-pick d2a569b
```

6. git bisect:

Discovering a recent bug but unsure when it was introduced? git bisect allows you to perform a binary search. By marking commits as 'good' or 'bad', Git will guide you to the problematic commit.

Examples:

```
$ git bisect start
$ git bisect bad          # Mark the current state as bad
$ git bisect good b3f5c8a # Mark the last known good commit
# Git will checkout a commit halfway between b3f5c8a and HEAD for testing
# If the midpoint commit is bad:
$ git bisect bad
# If the midpoint commit is good:
$ git bisect good
# Repeat until Git pinpoints the faulty commit.

$ git bisect reset        # End the bisect session
```

7. git fetch --prune:

If a colleague deletes a branch on the remote repository, your local list might still show it. To synchronize and remove references to stale branches, use git fetch --prune.

Examples:

```
$ git fetch --prune
```

Conclusions:

Beyond the foundational Git commands lies a world of utilities waiting to enhance your Git experience. By incorporating these commands into your daily routine, you'll leverage Git's full prowess, optimizing and streamlining your workflow.

References:

- Torvalds, L. (2012). The Git Parable.
- Chacon, S. & Straub, B. (2014). Pro Git. Advanced Git Log.
- Kernighan, B. & Ritchie, D. (1978). The C Programming Language.