Git - Cheat Sheet

To make it easier for us to look up and utilize the most important and commonly used commands when working with Git, the **cheat sheet** below is a helpful reference guide.

Setup

Setting up user data that is utilized in all local repositories.

```
git config --global user.name “[firstname lastname]”
```

Choose a name that clearly identifies us for attribution when reviewing version history.

```
git config --global user.email “[valid-email]”
```

Specify the email address that will be associated with every historical marker.

```
git config --global color.ui auto
```

For easier reviewing, configure Git to automatically color commands in the command line.

Setup and Initialization

Setting up user details, initializing repositories, and cloning repositories.

```
git init
```

Set up an existing directory as a repository for Git.

```
git clone [url]
```

Using a URL, retrieve the complete repository from a hosted location.

Stage and Snapshot

Using the Git staging area and snapshots.

```
git status
```

Display the updated files in our working directory, ready for our next commit.

```
git add [file]
```

Add the current version of the file to our upcoming commit (stage).

```
git reset [file]
```

Unstage a file while keeping the working directory's changes.

```
git diff
```

View the changes that have been staged.

```
git diff --staged
```

Diff of what is staged but not yet commited.

```
git commit -m "[descriptive message]"
```

Make a fresh commit snapshot with our staged content.

Branch and Merge

Isolating work in branches, changing context, and integrating changes.

```
git branch
```

List our branches, the branch that is currently active will have a * next to it.

```
git branch [branch-name]
```

Create a new branch after the most recent commit.

```
git checkout
```

Switch to a different branch and check it out into our working directory.

```
git merge [branch]
```

Merge the history of the specific branch with the current one.

```
git log
```

Show the history of all commits made to the current branch.

Inspect and Compare

Examining logs, diffs, and object data

```
git log
```

Show the history of all commits made to the current branch.

```
git log branchB..branchA
```

Display the branch A commits that are not on branch B.

```
git log --follow [file]
```

Display the changes made by commits to the file, even after renaming.

```
git diff branchB...branchA
```

Display the differences between what is in branch A and what is not in branch B.

```
git show [SHA]
```

Show any object in Git in human-readable format.

Tracking Path Changes

Versioning file removals and path modifications

```
git rm [file]
```

Remove the file from the project and prepare it for commit.

```
git mv [existing-path] [new-path]
```

Change an existing file path and stage the move.

```
git log --stat -M
```

Display all commit logs along with indication of any paths that moved.

Ignoring Patterns

Preventing files from being accidentally staged or committed.

```
logs/
*.notes
pattern*/
```

Save the desired patterns in a file ending in **.gitignore** and use either wildcard globs or direct string matches.

```
git config --global core.excludesfile [file]
```

Ignore pattern for all local repositories across the system.

Share and Update

Retrieving updates from another repository and updating local repository.

```
git remote add [alias] [url]
```

Create an alias for a git URL.

```
git fetch [alias]
```

Fetch down every branch on that Git remote.

```
git merge [alias]/[branch]
```

In order to update our branch, merge a remote branch into it.

```
git push [alias] [branch]
```

Transfer commits from the local branch to the remote repository branch.

```
git pull
```

Fetch commits from the tracking remote branch and merge them.

Rewrite History

Changing commits, rewriting branches, and deleting history

```
git rebase [branch]
```

Apply any current branch commits before the designated one.

```
git reset --hard [commit]
```

Apply any current branch commits before the designated one.

Temporary Commits

Store tracked files that have been updated temporarily in order to switch branches.

```
git stash
```

Save modified and staged changes

```
git stash list
```

List stack-order changes for stored files

```
git stash pop
```

Work from the top of the stash stack when writing

```
git stash drop
```

Remove the modifications from the top of the stash stack.