Git Cheat Sheet

Initializing a Repository

- **Initialize a new Git repository**:
  `git init`
- **Stop tracking Git repository**:
  `rm -rf .git` (removes the `.git` directory)

Viewing Changes

- **See the current status of files** (modified/untracked):
  `git status`
- **See differences between changes**:
  `git diff`

.gitignore

- **Create a `.gitignore` file** to list files/folders you want Git to ignore.

Three Stages in Git

1. **Working Directory**: Contains untracked or modified files.
2. **Staging Area**: Organize files to commit in batches.
3. **.git Directory**: Stores repository metadata and history.

Staging Changes

- **Add all files to the staging area**:
  `git add -A`
- **See what is staged**:
  `git status`
- **Remove a specific file from staging area**:
  `git reset <file.js>`

- **Remove all files from staging area**:
  `git reset`

Committing Changes

- **Commit changes with a message**:
  `git commit -m "Message"`
- **Check commit log**:
  `git log`

Tracking an Existing Repository

- **Clone an existing repository**:
  `git clone <url> <directory_path>` (use `.` for the current directory)
- **List remote repositories**:
  `git remote -v`

Branches

- **List all branches (local and remote)**:
  `git branch -a`
- **Create a new branch**:
  `git branch <branch-name>`
- **Switch to a branch**:
  `git checkout <branch-name>`
- **Create and switch to a new branch**:
  `git checkout -b <branch-name>`
- **Delete a branch locally**:
  `git branch -d <branch-name>`
- **Delete a branch on the remote**:
  `git push origin --delete <branch-name>`

Pushing and Pulling Changes

- **Pull latest changes from a branch**:
  `git pull origin <branch>`

- **Push changes to remote repository**:
  `git push origin <branch>`
- **Associate local branch with remote branch** (first time only):
  `git push -u origin <branch-name>`

Merging Branches

- **Switch to master branch**:
  `git checkout master`
- **Merge another branch into the current branch**:
  `git merge <branch-name>`
- **Push merged changes to remote**:
  `git push origin master`
- **Check if a branch has been merged**:
  `git branch --merged`

Example Workflow

1. **Create and switch to a new branch**:
   `git branch <branch-name>`
   `git checkout <branch-name>`
2. **Make changes, then stage and commit**:
   `git add -A`
   `git commit -m "Description"`
3. **Push branch to remote and set tracking**:
   `git push -u origin <branch-name>`
4. **Ensure master branch is up-to-date**:
   `git checkout master`
   `git pull origin master`
5. **Merge branch into master**:
   `git merge <branch-name>`
   `git push origin master`
6. **Delete the branch locally and remotely if no longer needed**:
   `git branch -d <branch-name>`
   `git push origin --delete <branch-name>`

Git Cheatsheet

Making and Unstaging Changes

- **Stage changes** (after modifying `file.js`):
  `git add file.js`
- **Unstage changes** (if you decide not to commit yet):
  `git restore --staged file.js`

Restoring Deleted Files

- **Restore a file accidentally deleted**:
  `git restore file.js`

Viewing Commit History

- **View all changes made in each commit**:
  `git log -p`
  (`-p` shows the diff for each commit)

Going Back to a Previous Commit

- **Revert to an earlier commit**:
  `git reset <commit-code>`
  (where `<commit-code>` is the commit hash found in `git log`)

Rebasing Commits

Git rebase is used to reapply commits on top of another base tip, typically to clean up or reorganize commit history. Here are some common rebase commands:

1. **Interactive Rebase from the Root Commit**
   `git rebase -i --root`
   This command opens an interactive rebase session from the very first commit, allowing you to rewrite or organize every commit in the repository history.
2. **Rebase from a Specific Commit**
   `git rebase -i <commit-code>`
   Start an interactive rebase from a specific commit onward. This is helpful if you want to edit, squash, or reorganize recent commits.

3. **Rebase to Update with a Remote Branch**
   `git rebase origin/master`
   Instead of merging, this will replay your branch's commits on top of the latest commits in `origin/master`. This keeps the commit history linear.

Rebase Options in Interactive Mode

When running `git rebase -i`, you'll enter a text editor with a list of commits and options to manage them:

- **Pick**: Keeps the commit as-is.
- **Edit**: Allows you to make changes to the commit, including its message or staged content.
- **Squash (s)**: Combines this commit with the one directly before it, useful for reducing multiple commits into one.
- **Fixup (f)**: Similar to `squash` but discards the commit message; combines it with the previous commit.
- **Drop**: Removes the commit from history entirely.

Example Rebase Workflow

1. **Start an interactive rebase**:
   `git rebase -i HEAD~3`
   (This will open an editor for the last 3 commits; adjust the number as needed)
2. **Choose options** (`pick`, `edit`, `squash`, etc.) for each commit, then save and close.

If you chose `edit`, make your changes and re-commit:

```
git add <file.js>
git commit --amend
git rebase --continue
```

3. **If you encounter conflicts**, resolve them, add changes (`git add <file.js>`), and continue rebasing with:
   `git rebase --continue`

Example Workflow for Fixing Commit History

1. **Check commit history**:
   `git log`
2. **Rebase interactively to organize commits**:
   `git rebase -i HEAD~4`
   (for the last 4 commits, or adjust as needed)
3. **Combine or remove commits as needed** using `squash`, `edit`, or `drop` options.
4. **Complete the rebase**:
   Resolve conflicts if they arise, then use `git rebase --continue`.

To make sure code is up to date,
git checkout main
git pull
git checkout YOUR_BRANCH_HERE
git rebase main