# git

// Medhat Riad


CHECK-IN

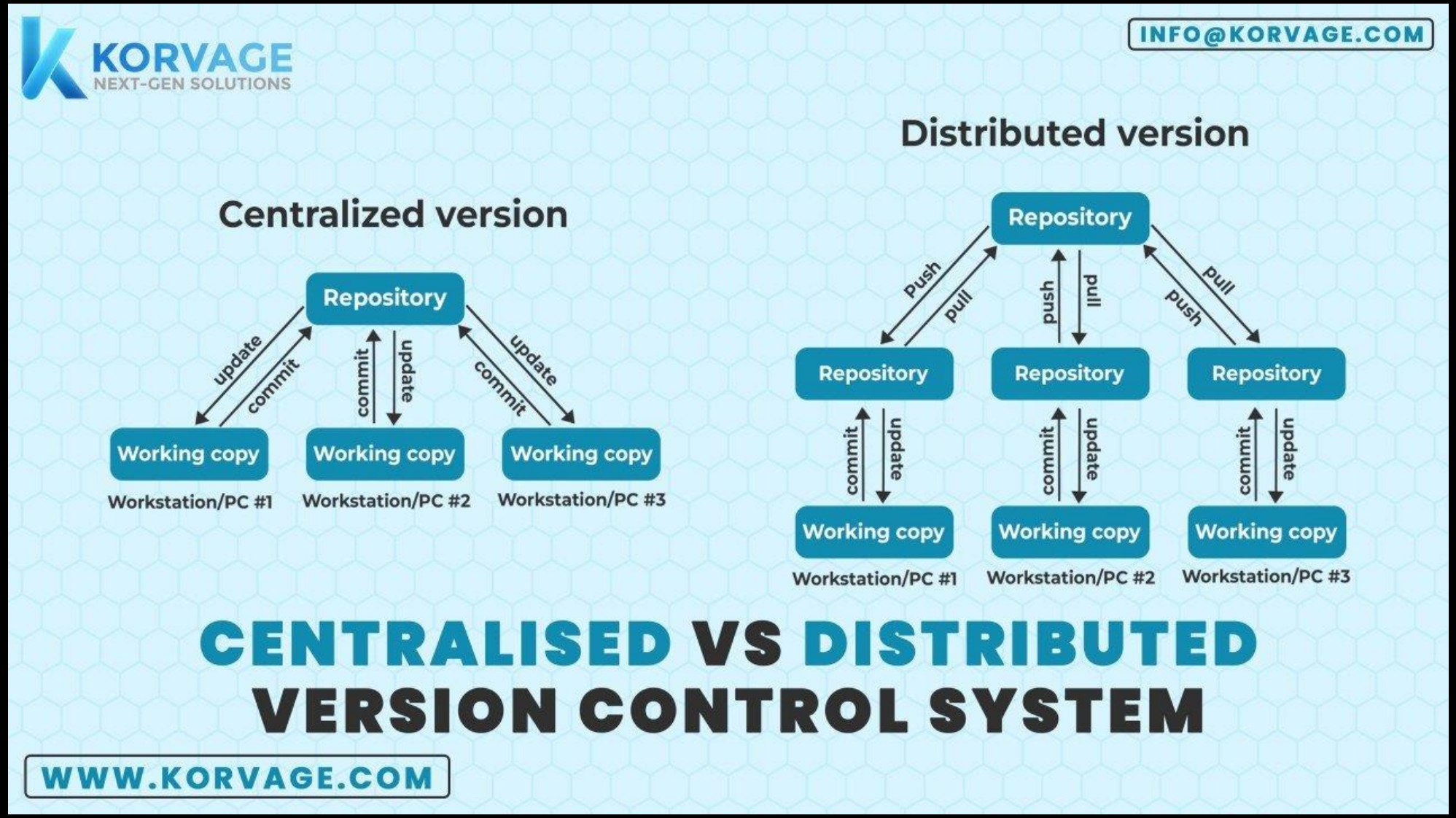
# THE FEAR OF GIT


# VERSION CONTROL

- Snapshots of your code over time
- Allows different members to work on different parts of the code simultaneously
- Rewind capabilities

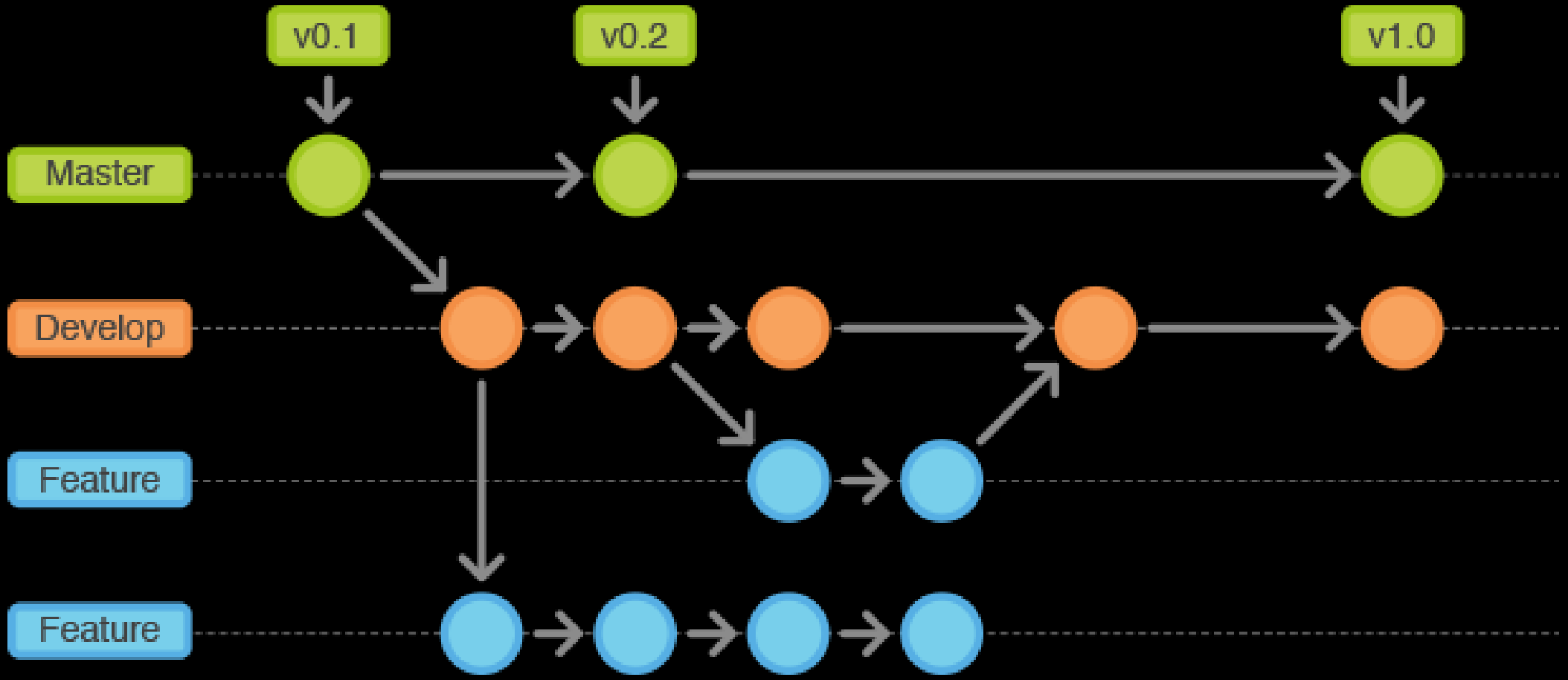

00:23.62
https://flynn-delta.itch.io/overdrive

# VERSION CONTROL





# BRANCHES AND COMMITS




DEMO

# GETTING STARTED

- Create a new repository
  ```
  > git init
  ```
- Clone an existing repository
  ```
  > git clone <url>
  ```

---

- Check status
  ```
  > git status
  ```
- Check log
  ```
  > git log
  ```

---

- Add file to be committed
  ```
  > git add <file>
  ```
- Add all files to be committed
  ```
  > git add .
  ```

---

- Commit added files
  ```
  > git commit –m ‘message’
  ```

---

- Create a new branch
  ```
  > git branch <branch-name>
  ```
- Switch to branch
  ```
  > git checkout <branch-name>
  ```


ATOMIC COMMITS

# USE MEANINGFUL COMMIT MESSAGES

# CONFLICT RESOLUTION

- Merging two branches may cause a conflict
- A conflict means **Git couldn’t choose for you**; it’s not an error, it’s a **question**.
- You resolve by **editing**, not by commands alone.
- After resolving, you **add** the file (mark resolved) and **commit** to complete the merge.


DEMO

# MERGING AND CONFLICTS

- Merge branch into current branch

```
> git merge <branch>
```

---

- Abort failed merge (conflicts)

```
> git merge --abort
```

- Continue merge after resolving conflicts

```
> git merge --continue
```

---

- Add resolved file to be committed

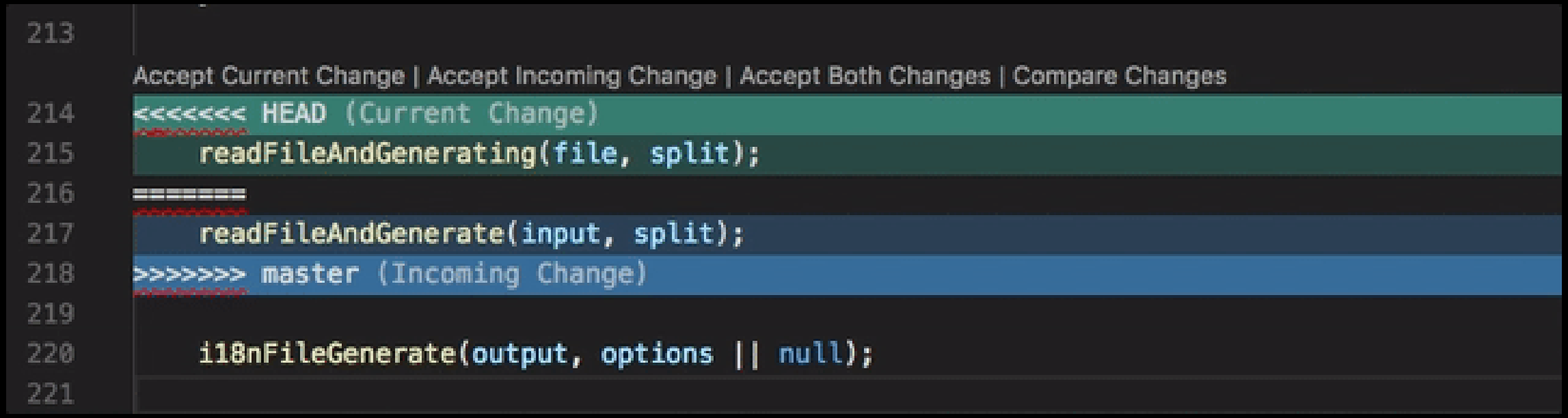
```
> git add <file>
```

---

- Commit added files

```
> git commit
```

# RESOLVING CONFLICTS IN VSCODE




J

# WORKFLOW EXAMPLE – SOLO DEV

1. Switch to main and get latest
   ```
   > git checkout main
   > git pull
   ```
2. Create a branch for your feature
   ```
   > git checkout -b feature/add-login
   ```
3. Make changes
   ```
   > git add .
   > git commit -m "Add login form”
   ```
4. Sync with remote repo
   ```
   > git push origin feature/add-login
   ```
5. Merge when done
   ```
   > git checkout main
   > git pull
   > git merge feature/add-login
   > git push
   ```
6. Clean up
   ```
   > git branch -d feature/add-login
   > git push origin --delete feature/add-login
   ```

# WORKFLOW EXAMPLE – TEAM N00BS

1. Switch to main and get latest
   ```
   > git checkout main
   > git pull
   ```
2. Create a branch for your feature
   ```
   > git checkout -b feature/add-login
   ```
3. Make changes
   ```
   > git add .
   > git commit -m "Add login form”
   ```
4. Stay up to date
   ```
   > git fetch origin
   > git rebase origin/main
   ```
5. Merge when done
   ```
   > git checkout main
   > git merge --ff-only feature/add-login
   > git push
   ```
6. Clean up
   ```
   > git branch -d feature/add-login
   ```

# WORKFLOW EXAMPLE – TEAM PROS

1. Switch to main and get latest
   ```
   > git checkout main
   > git pull
   ```
2. Create a branch for your feature
   ```
   > git checkout -b feature/add-login
   ```
3. Make changes
   ```
   > git add .
   > git commit -m "Add login form”
   ```
4. Sync with remote repo
   ```
   > git push origin feature/add-login
   ```
5. Open a Pull request (On GitHub/GitLab)
   - **Base branch:** main
   - **Compare branch:** feature/add-login
   - Add a **title, description**, and any context.
6. **Review & discuss**
   - Comment or suggest changes
   - Testing
   - Fix feedback and push more commits (they appear automatically in the PR)
7. If main has changed since you branched
   ```
   > git fetch origin
   > git merge origin/main
   > git push
   ```
8. **Merge**
   - **Squash merge** → one clean commit
   - **Rebase & merge** → keeps linear history
   - **Normal merge** → not recommended
9. **Clean up**
   ```
   > git branch -d feature/add-login
   > git push origin --delete feature/add-login
   ```


QUESTIONS

# FEEDBACK

# RESOURCES