Version Control System

Ranjit Karni

VPark Innovations
Ranjit.balu@gmail.com
Ph.No: 9676976662

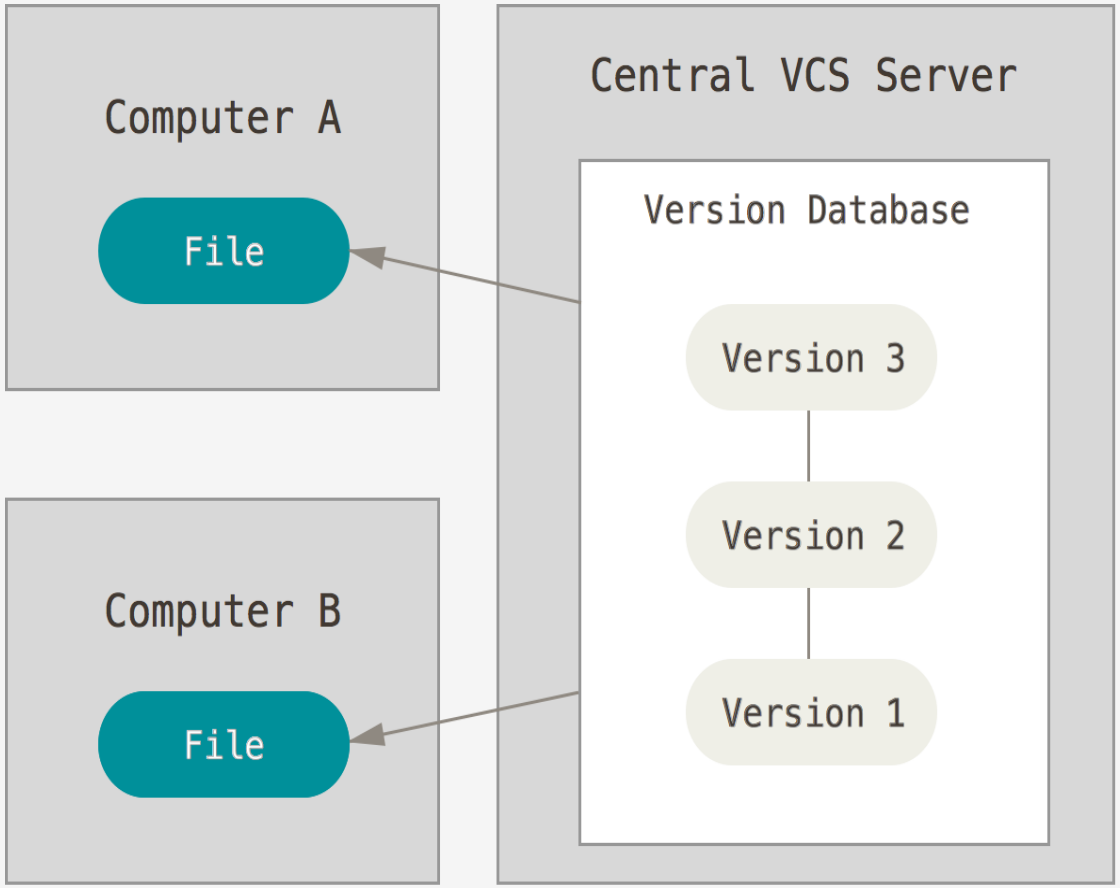
Agenda

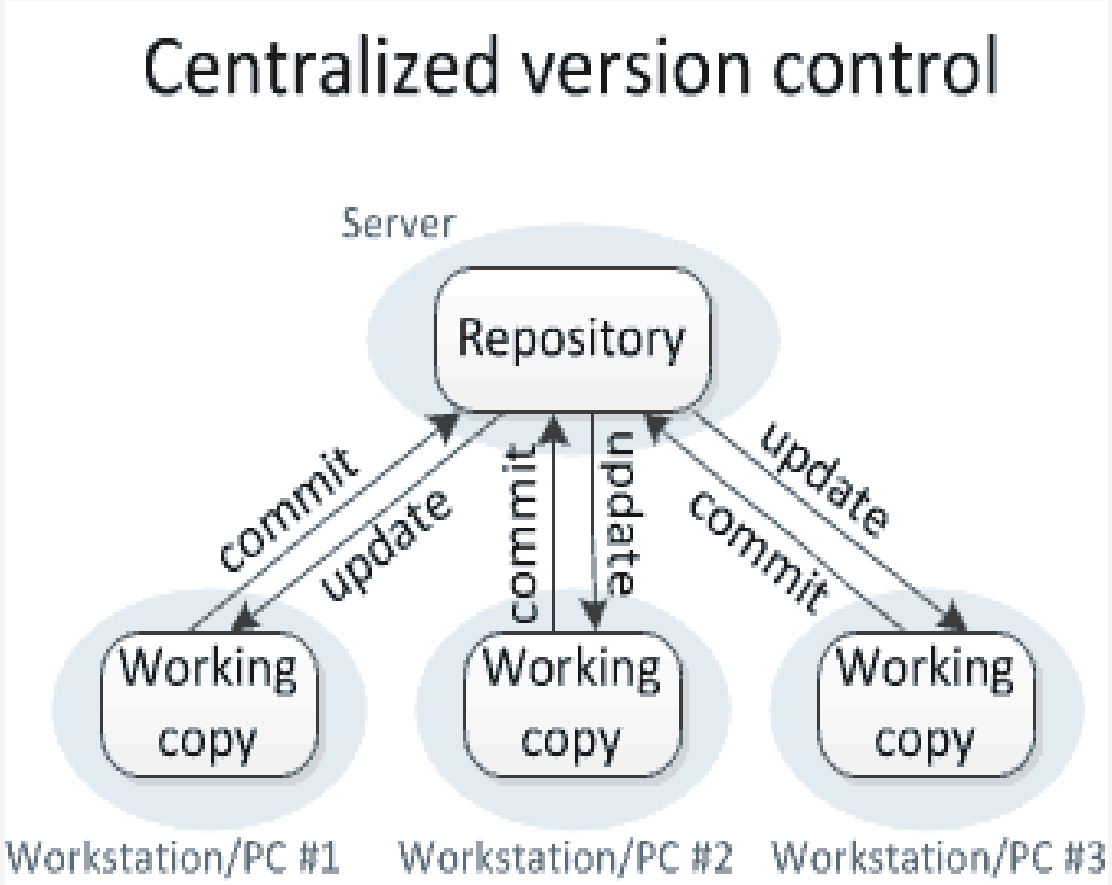
1. Version Control Systems
2. Git as a version control tool
3. Git setup and configuration
4. Basic flow: commit / push / pull
5. Branches and merging
6. Undoing changes
7. Troubleshooting techniques
8. Best practices

VERSION CONTROL SYSTEMS

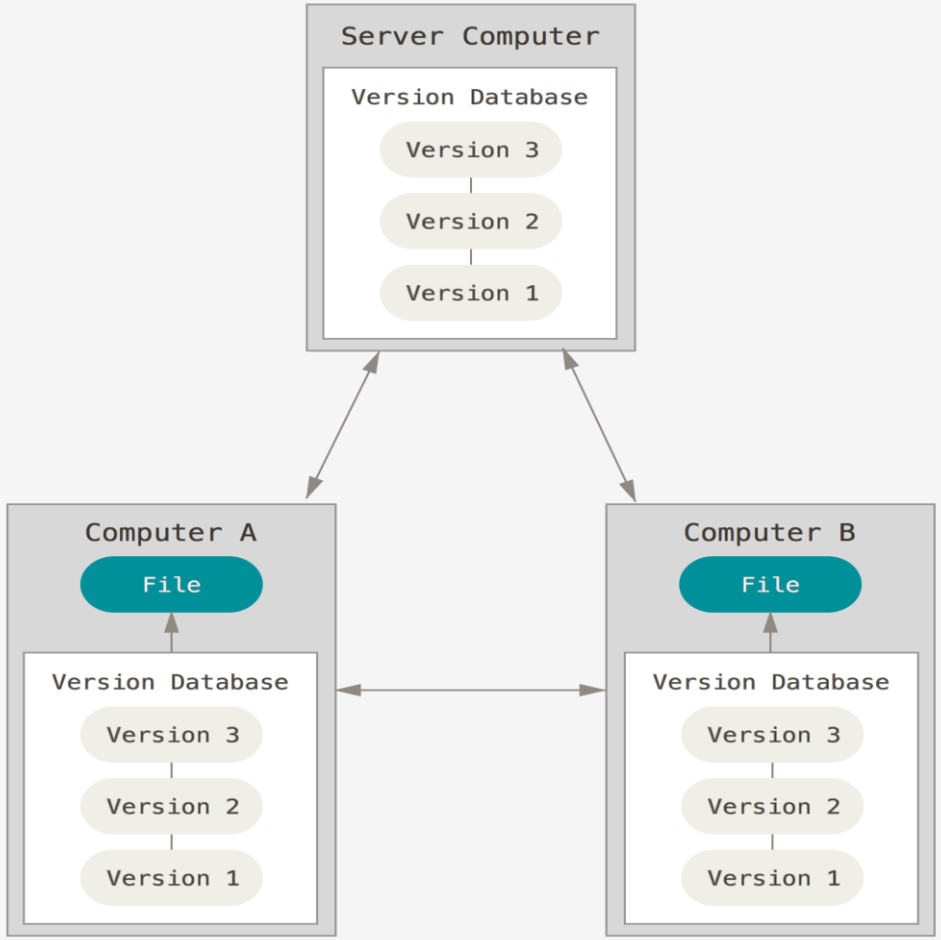
- **GOALS**
- Backup and Restore
- Synchronization
- Undo
- Track Changes
- Track Ownership
- Branching and merging

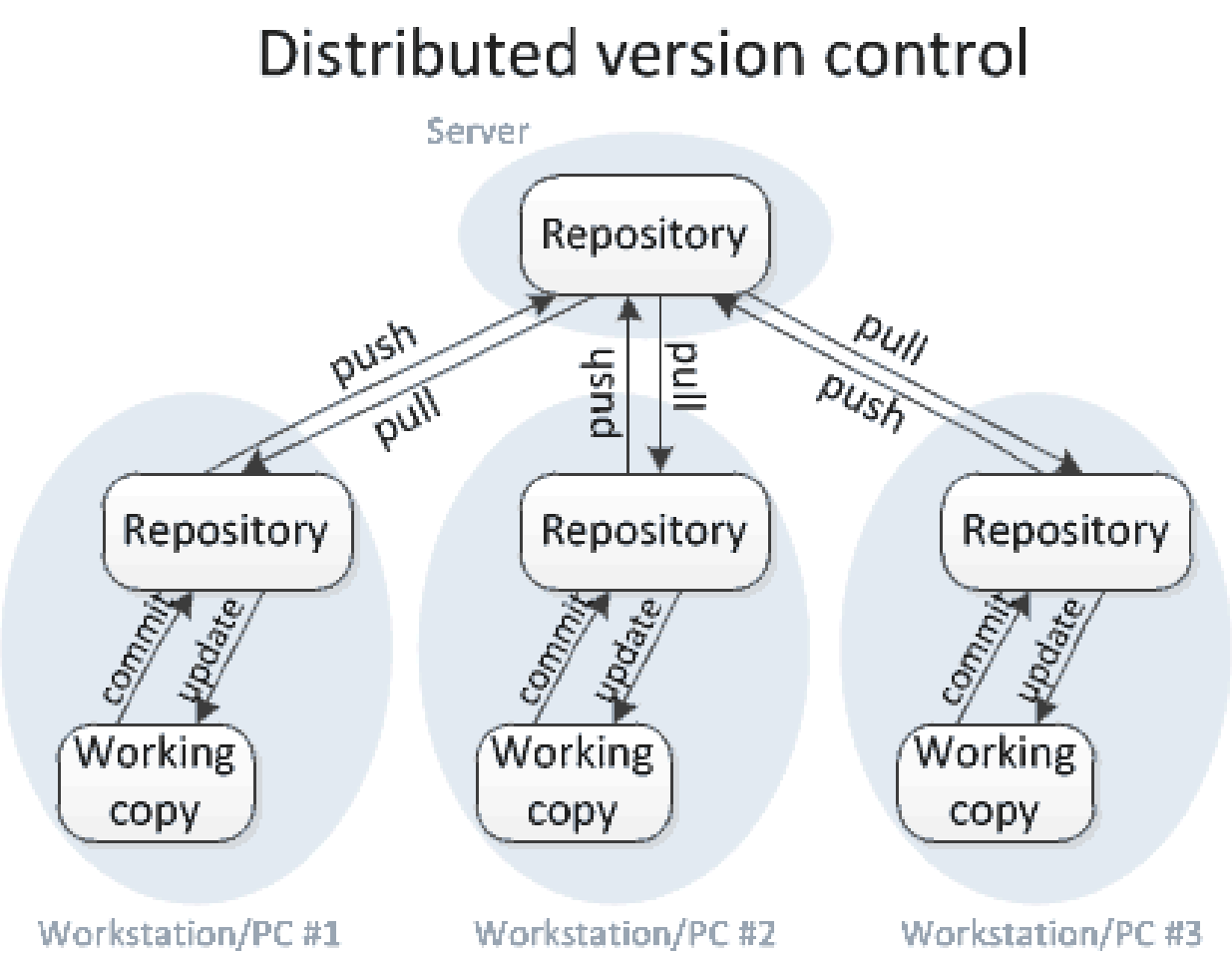
CENTRALIZED VS DISTRIBUTED





CENTRALIZED VS DISTRIBUTED





GIT vs SVN

- ***GIT is Fast – Almost all operations are local.***
- ***GIT is Distributed – Entire history is maintained in local.***
- ***Size of the working copy is comparatively large.***
- ***Code can be committed even if the server is down.***
- ***History can be restored even if the server crashes***

- SVN is Slower – On every operation, communication between working copy and server happens.
- SVN is Central – Only the current versions of files are stored in local.
- Size of the working copy is smaller
- Code cannot be committed until the server is up
- In case the server crashes, there is no way to restore the history.

GIT vs SVN

- ***GIT does not support Lock-Modify-Unlock concept***
- ***In GIT, entire repository needs to be cloned.***
- ***It is not preferred to commit binary files and large files in GIT***
- ***Adding/ Committing code to GIT involves double the no of commands***
- ***GIT repository is either public or private (Paid Service)***

- SVN supports Lock-Modify-Unlock concept as well
- In SVN, we can checkout the sub-folders.
- SVN provides better support for binary and large files
- Adding/Committing code to SVN involves considerably fewer steps
- SVN repository is private and free – More secure

GIT

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency

WHY GIT

- Fast
- Distributed
- Data Assurance
- Free and Open Source

GIT SETUP

Download binary from here: http://git-scm.com/downloads
Follow the steps using the default options

Configure your identity

- Generate SSH key pair
  - `ssh-keygen -t rsa -C "your_email@example.com"`
- Send public key to repository owner or upload to your profile
- Configure username and email
  - `git config --global user.name "YOUR NAME“`
  - `git config --global user.email "YOUR EMAIL"`

GIT BASE FLOW DEMO

Clone Repository

git clone <url> [--branch <branchname>] [<foldername>]

Checkout/change branch

git checkout <branchname>

Pull Remote Repository

git pull

Initialize the new git repository

git init

Add file to Staging

git add ./-A : Adds all files (Modified + Newly added)

git add –u : Adds only Modified files

git add <filename> : Adds particular file

GIT BASE FLOW DEMO

Commit to Staging

```
git commit –m “<commit message>”
```

Push to remote repository

```
git push
```

Check logs

```
git log
```

Create new branch

```
git branch origin <newbranchname> <oldbranchname>
```

Push branch to remote

```
git push –set-upstream origin <newbranchname>
```

UNDOING CHANGES

Working directory

- `git checkout -- file.txt`
- `git checkout .`
- `git clean -xdf`

Staging area (Index)

- `git reset -- file.txt`

Local branch

- `git reset HEAD^^ (HEAD~2)`
- `git commit --amend -m “commit message”`

Remote repository

- `git revert <sha1>`

GIT RESET

Soft

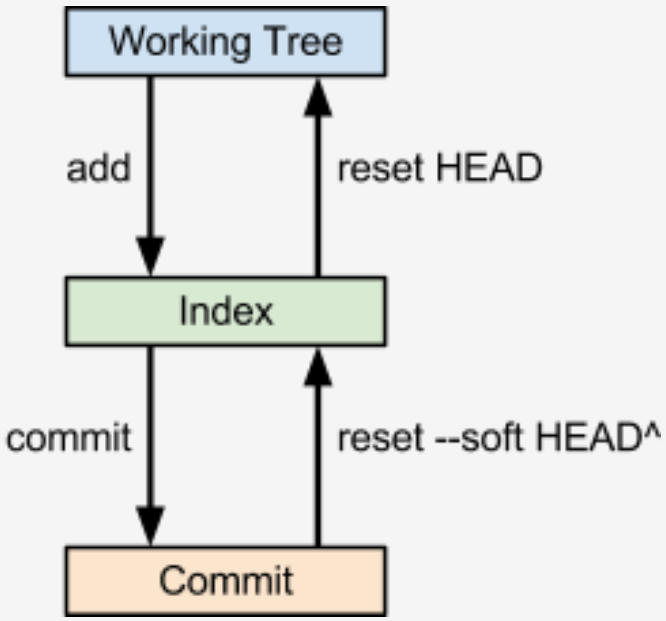
- Change HEAD to old SHA1
- `git reset --soft <sha1>`
- Add changes into index

Mixed (default)

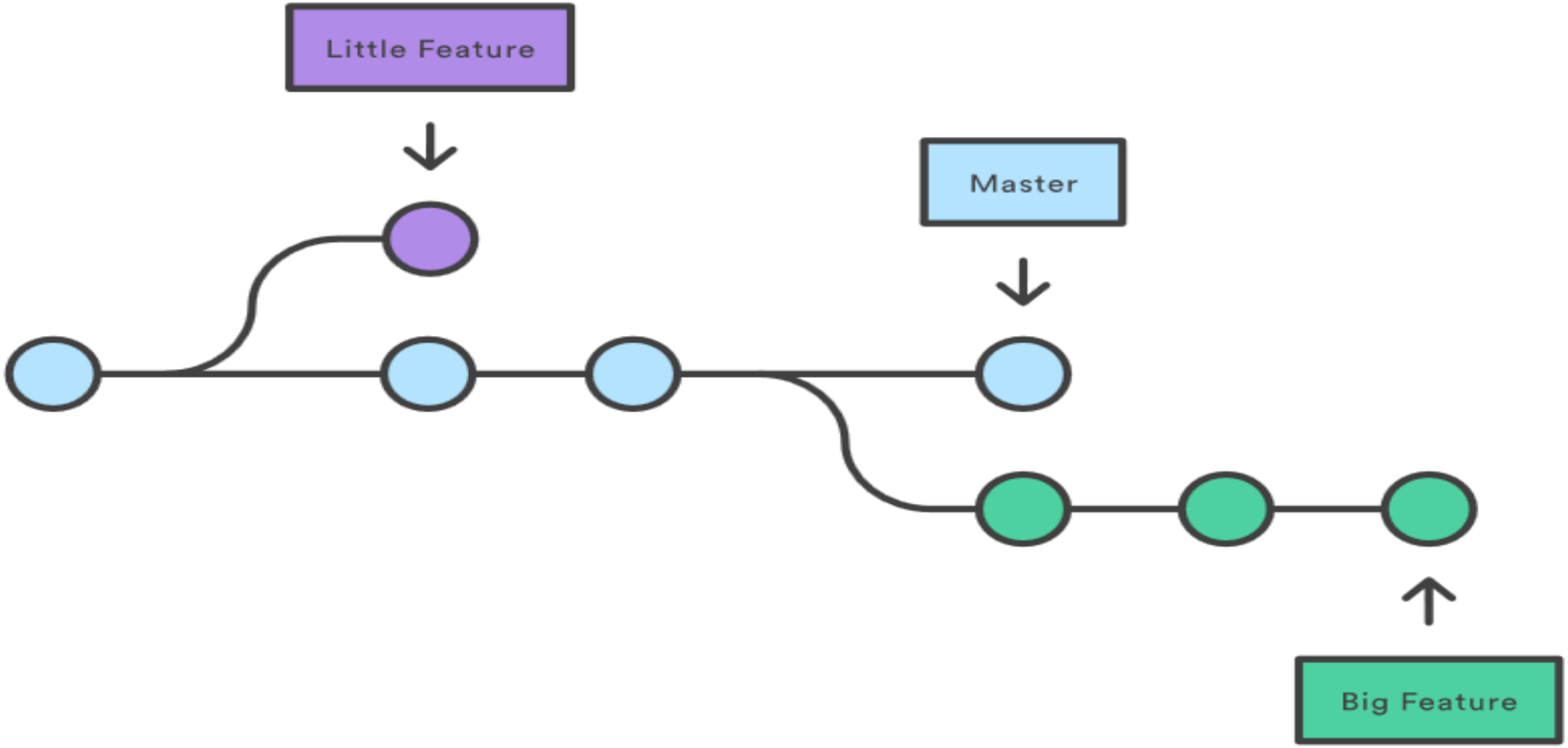
- Change HEAD to old SHA1
- `git reset -- mixed <sha1>`
- Do not add changes to index

Hard

- Change HEAD to old SHA1
- `git reset -- hard <sha1>`
- Do not saves changes.
- Do not remove the new commit, but git gc can)



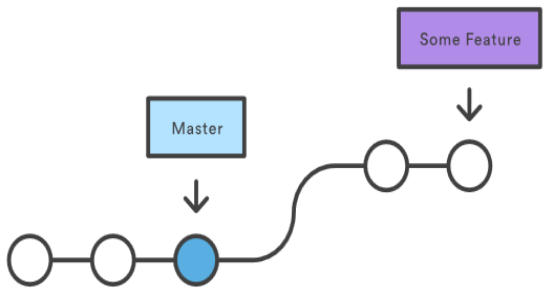
BRANCHING

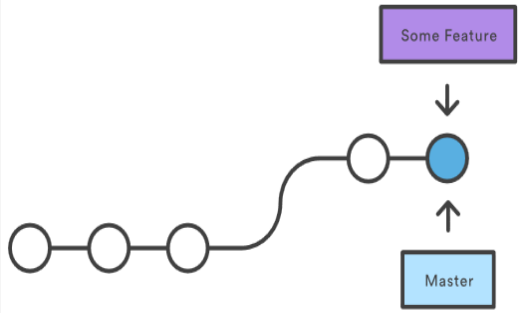


MERGING

Fast-forward merge

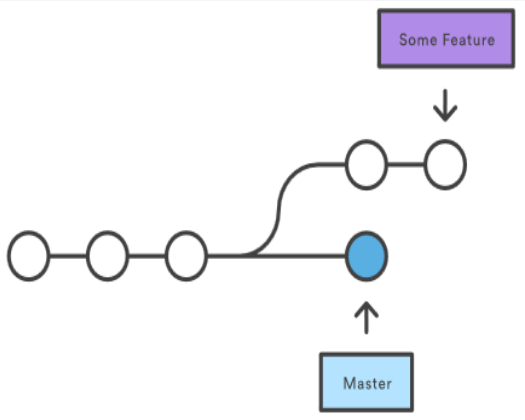
Do not produce additional commit

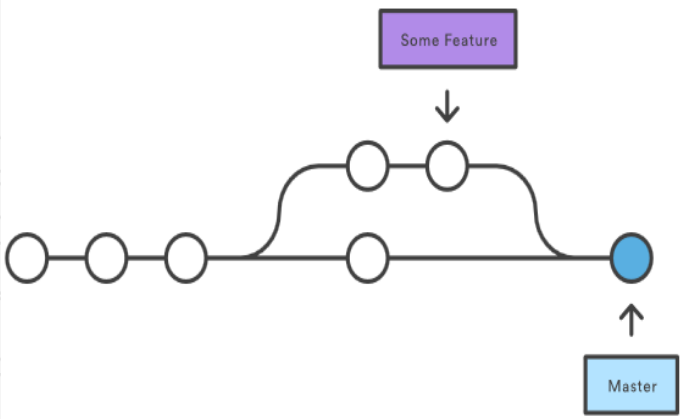




Non-ff merge

Produce additional commit with details





CONFLICTS

Abort merge

```
git merge –abort
```

Resolve by selecting version

```
git checkout --ours –theirs
```

Resolve manually

```
git diff
```

User merge tool

Avoid conflicts

- Short commits
- No edits to whitespaces
- Merge often

.gitIgnore

Create .gitIgnore File

Go to .gitIgnore File -> Mention file in the format (Filename/*) -> Save

Commit The .gitIgnore File

Stashing

Stash

Go to Repository -> Git Bash -> Stash Save

Get the Stash list

Go to Repository -> Git Bash -> Stash List

Stash Apply

Go to Repository -> Git Bash -> Stash Apply <StashName>

Stash Drop

Go to Repository -> Git Bash -> Stash Drop <StashName>

Stash Pop(Stash Apply + Stash Drop)

Go to Repository -> Git Bash -> Stash Pop <StashName>

Merge Request

Create New Merge Request

Go to Git lab -> Merge Request -> Create New Request -> Enter Details (Title, assignee, etc) -> Ok

Merge the Code

Go to Git lab -> Merge Request -> Add Comment -> Uncheck Delete Source branch -> Merge

Resolve Conflicts If any

Troubleshooting techniques

1. File annotation:

git blame [-L 12,22] [-C] <filename.txt>

2. Binary Search:

```
git bisect start
git bisect bad
git bisect good v1.0


git bisect reset
```

GIT BEST PRACTICES

Identify yourself

Set username
Set email

Pull before push

Save you from unnecessary merges

Use default tools

If you are new to git

Do not commit garbage

Git is not the best place for logs, user settings and output binaries

Use branches

Branch is 41 bytes of memory

Questions?

Ranjit Karni

Vpark Innovations

Ranjit.balu@gmail.com

Ph No: +91-9676976662

Q&A

Thank you!

- For coming today
- For asking question

Further reading:

[Pro Git]

[Version Control with Git]