

- Git – Quick Start
- Building Your First Repository
- Working in a Team
- Advanced Features

- Keeping history of working versions of your projects
- know who made changes as well as when and why
- Merge Conflicting changes

# How Git approaches version control

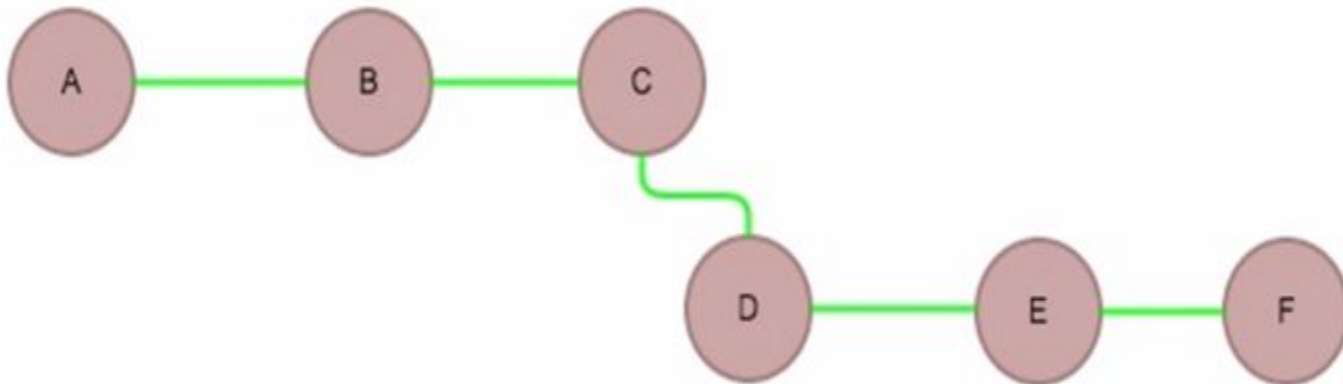
- Each commit is a snapshot of entire project not just the changes made
- Developers works on clones of the repository with complete access to history

# Advantages of distributed version control

- Multiple clones means multiple backups
- Ability to work offline
- Setup multiple independent workstation

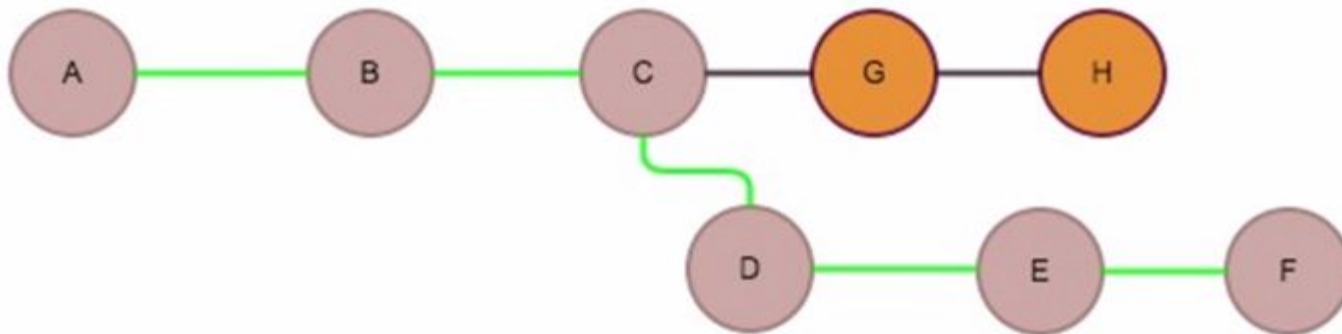
- Centralized Workflow
- Feature branch workflow
- Forking workflow

- It is similar to CVCS repository
- Team members all work off a single branch in a central repository
- Rebase unpushed commits on top of updates to the master branch
- Requires frequent synchronization on larger team
- It is suitable for smaller team but in larger team it required frequent synchronization to keep divergence manageable



# Tom's Diverged Branch

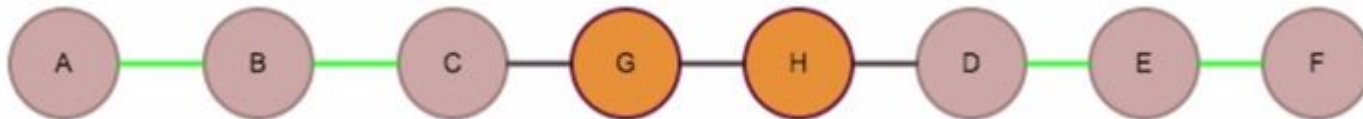
Branch get diverged cause of bob's commit





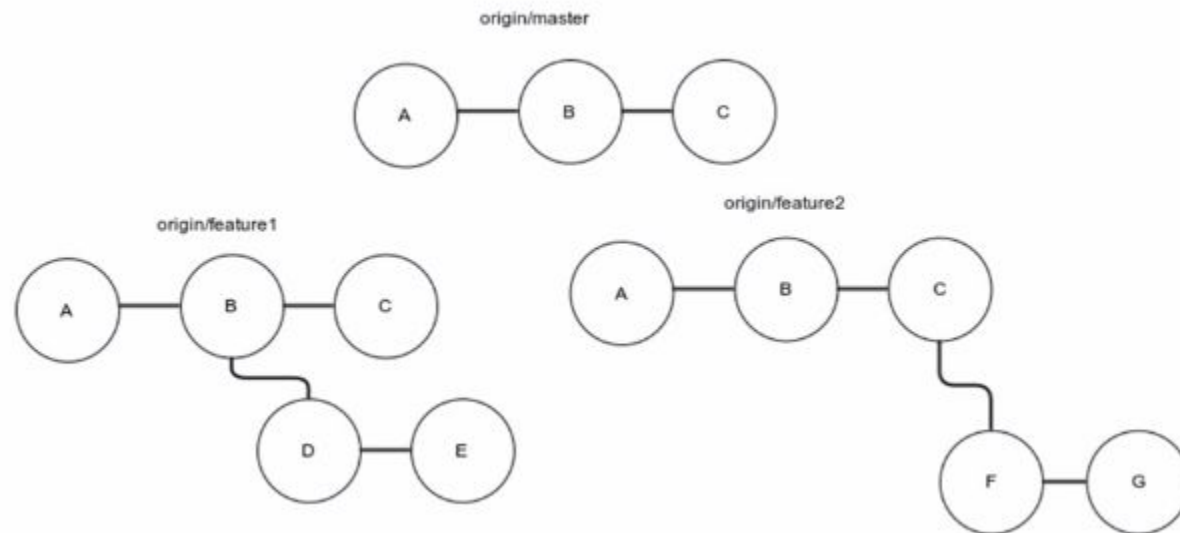
# Tom's Rebased Branch

Tom needs to incorporate Bob's changes



- Feature developed in dedicated branches
- Issue pull request from feature branch to master
- Keep main code base clean and working during development

# Feature Branch Workflow



- Each developer has dedicated online repository
- Write access is restricted to administrators
- Pull request are issued to main repository
- Every fork has information about its master branch
- It can track divergence between different repositories
- In this type of workflow the write-access is restricted to the maintainers only
- Here developers push commits to their own forks and make pull requests to the central repository
- Admin view these changes and decides if the changes can be merged

Different team dynamics requires different collaboration techniques.  
Choose the workflow that suits your group.

- Choose the familiarity of a centralized workflow while capitalizing on Git's additional advantages
- Create feature branches to keep new development from interfering with the stable code base
- Use the open source forking model to allow collaborators from anywhere while keeping your main repo protected

- Repository Initialization
- Making your first commit
- Managing remotes
- Viewing commit history

It is important to maintain logs:

- Initializing a local repository
- Tracing new files
- Example
  - Let meet a friend tom who has an idea for app
  - To starts developing his idea into reality with his programming skills
  - He takes the support of open source community for this purpose

1. `yum install git`
2. `mkdir -p /opt/gitdemo`
3. `cd /opt/gitdemo`
4. `git init`
5. `ls -a`
6. `vim problem_solver.rb`
7. `git status`
8. `git add problem_solver.rb`
9. `git status`

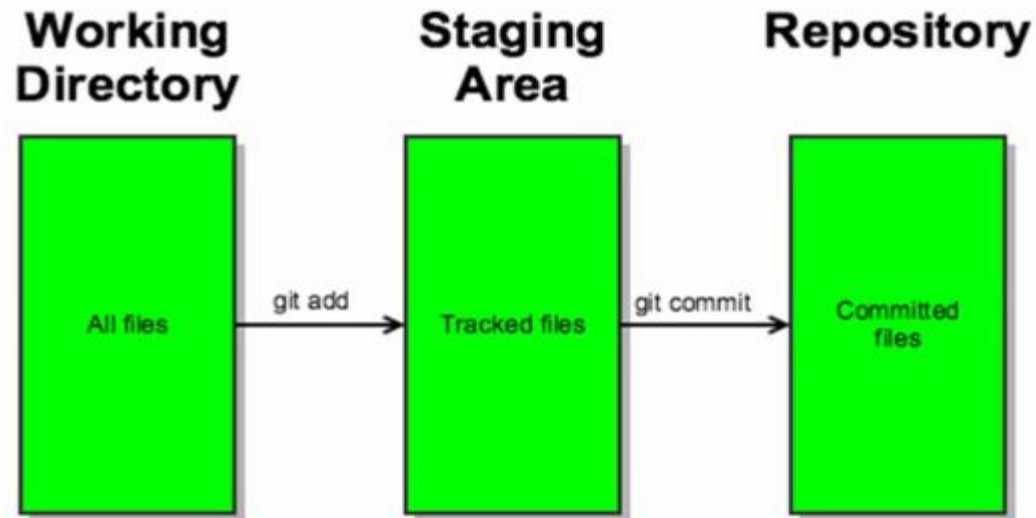


- Make changes in working directory
- Move changes to staging area
- Commit the changes to the repository
- README.md
  - ☐ What it does exactly
  - ☐ System requirements
  - ☐ Installation and running instructions
  - ☐ How to contribute to the project
  - ☐ md stands for **markdown** which is a simple syntax for providing semantic information and representing common formatting in plain text
- .gitignore is a hidden file which lists the files to ignored while committing

# The First Commit (Commands)

1. vim problem\_solver.rb
  - I. class Problem Solver
  - II. def solve\_easy\_problem
  - III. end
  - IV. end
2. vim README.md
  - I. Install Ruby
3. Vim .gitignore
  - I. \*.swp
4. gt status
5. git diff problem\_solver.rb
6. git config --global user.mail "mahesh.s.kharwadkar@gmail.com"
7. git config --global user.name "maheshkharwadkar"
8. git add .
9. git commit -m "first commit"

# The First Commit

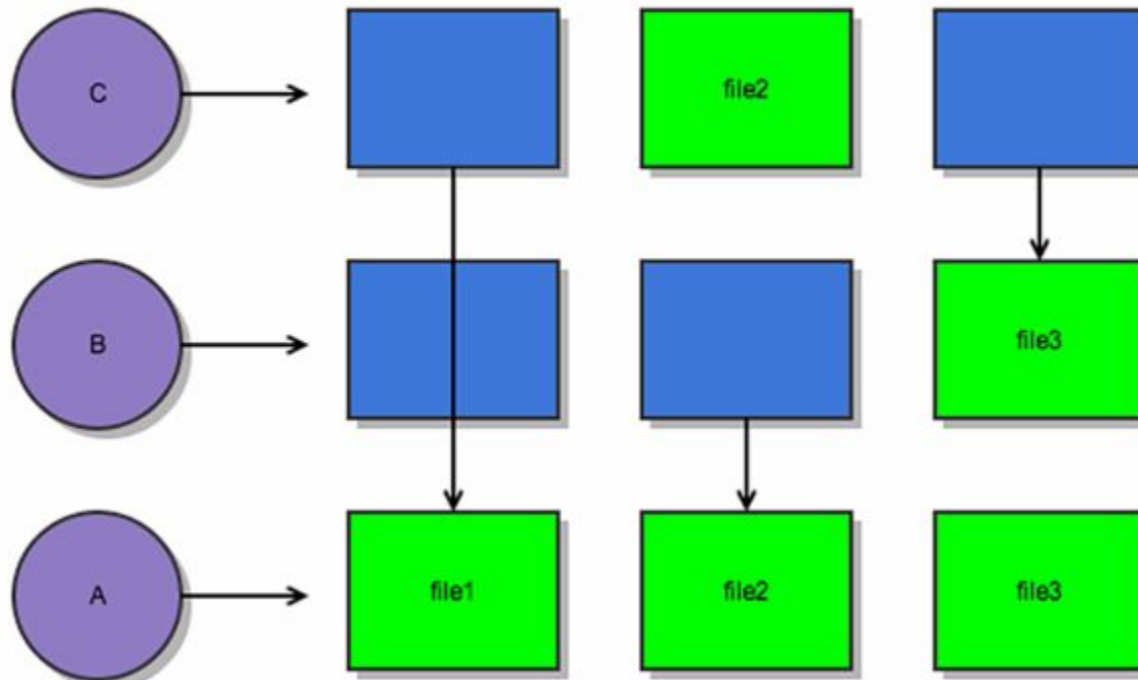


# The First Commit (Commands)

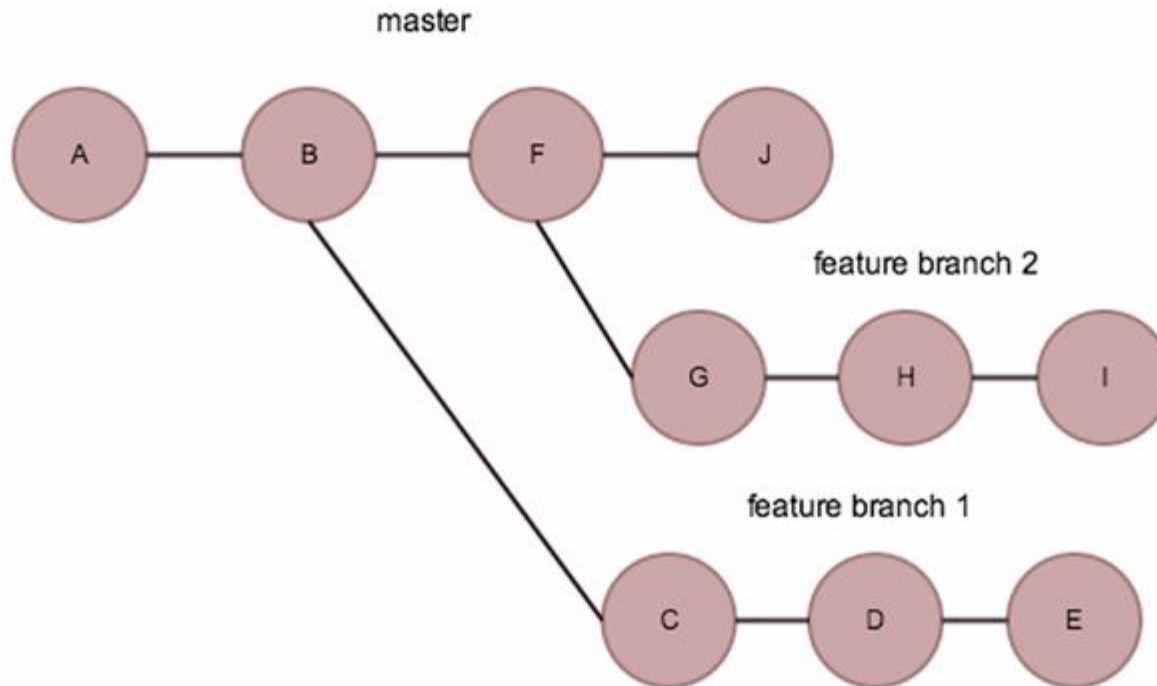
1. vim problem\_solver.rb
  - I. class Problem Solver
  - II. def solve\_easy\_problem
  - III. end
  - IV. def solve\_hard\_problem
  - V. end
  - VI. end
2. git add .
3. git diff
4. vim problem\_solver.rb
  - I. class Problem Solver
  - II. def solve\_easy\_problem
  - III. end
  - IV. def solve\_hard\_problem
  - V. end
  - VI. def solve\_tougher\_problem
  - VII. end
  - VIII. end
5. git diff **HEAD**
6. git diff

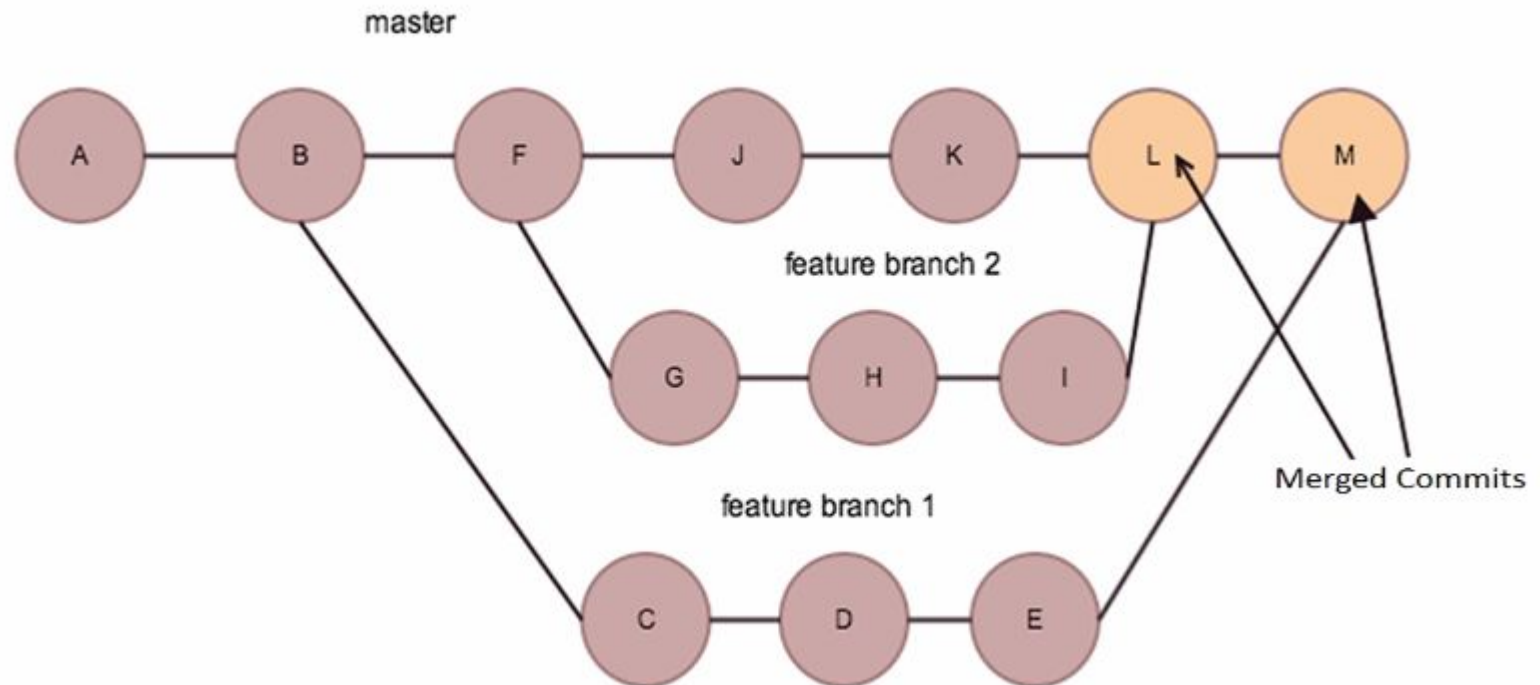
- What a Remote is
  - A copy of your repository hosted online
  - Makes your repository available for collaboration
  - E.g. Github, Bitbucket, beanstalk and codebase
- Adding a remote and pushing changes
  - Create your repository in github.com
    1. `git remote add origin https://github.com/maheshkharwadkar/gitdemo.git`
    2. `git remote`
    3. `git push origin master`
- Authentication options
  1. `ssh-keygen -t rsa -C "mahesh.s.kharwadkar@gmail.com"`
  2. `cat ~/.ssh/id_rsa.pub`
  3. copy ssh key to git hub -> settings -> New SSH key
  4. `git remote set-url origin git@github.com:maheshkharwadkar/gitdemo.git`
  5. `touch new_file.txt`
  6. `git add new_file.txt`
  7. `git commit -m "new blank file"`
  8. `git push origin master`

- Origin/master – points to tip of master branch on remote
- HEAD – points to tip of current local branch
- git log command used to see the history
- Use Commit hash to view the state of the project at any point in history



- Origin/master – points to tip of master branch on remote
- HEAD – points to tip of current local branch
- git log command used to see the history
- Use Commit hash to view the state of the project at any point in history

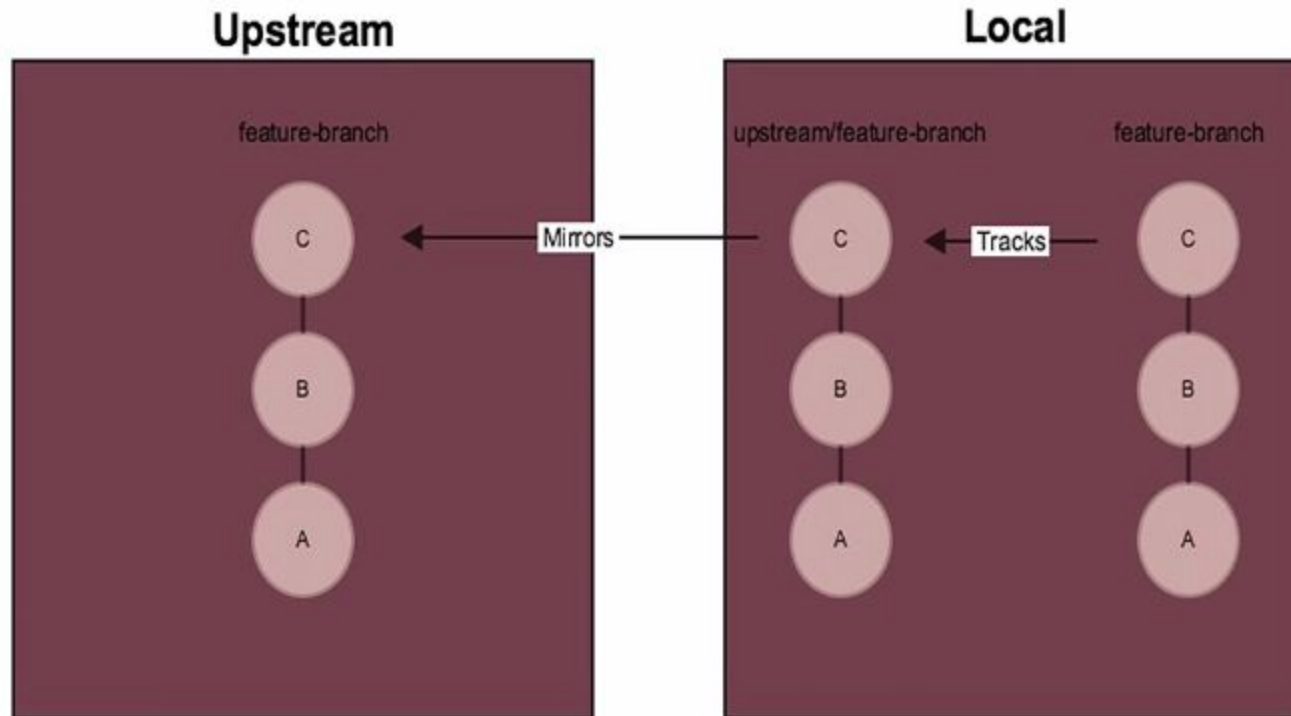


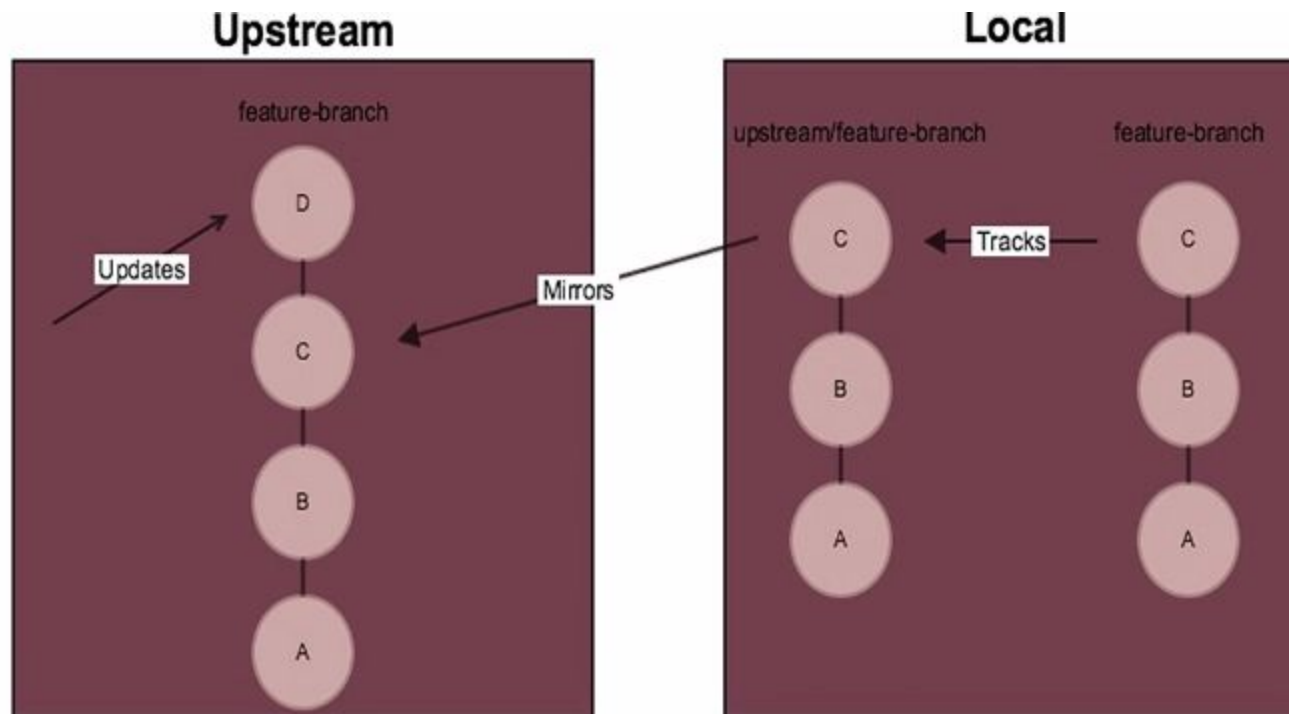


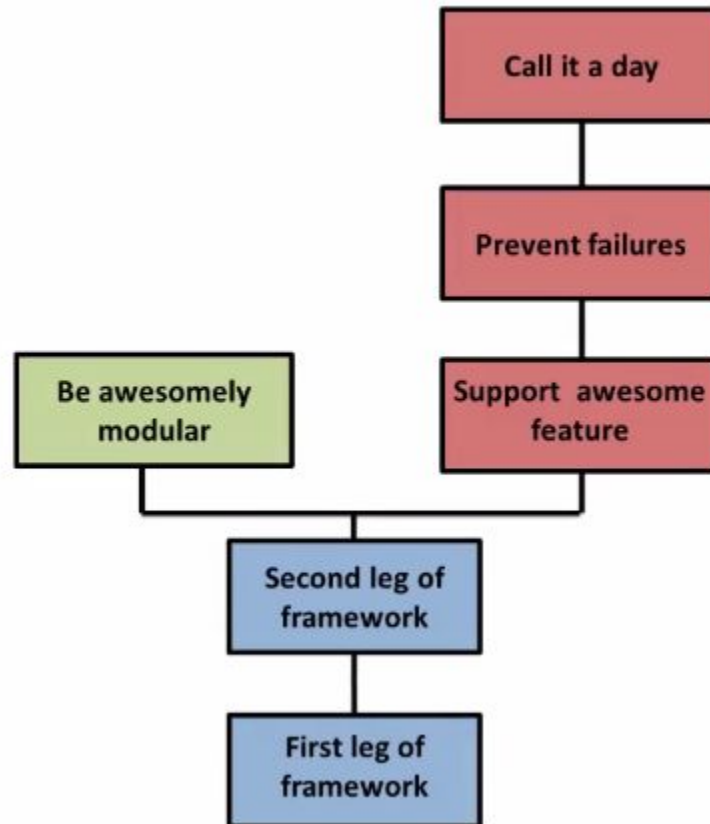


- Till now, Zac has created a repository, made a few commits and pushed them to Github
- He is now looking for a new collaborator to implement a new feature in his app
- He finds an interested contributor named Sara
- Sara needs to find a place to submit her work
- Zac creates a new branch to isolate his work from those of the contributors to the master branch; he pushes this branch upstream
- Sara's work will be stored here before it merges with the master branch

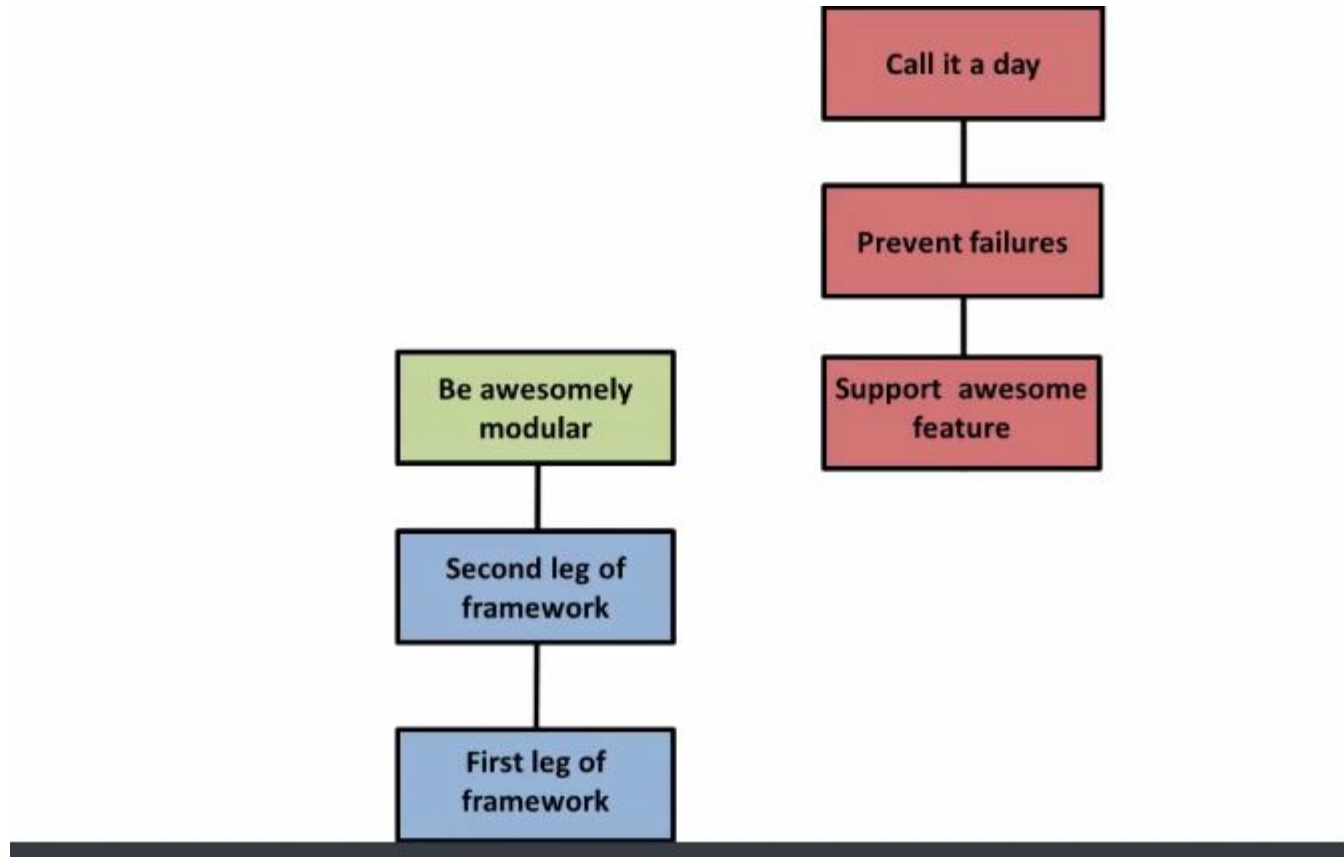
- --Branching Example
  - `git branch my-awesome-feature`
  - `git push origin my-awesome-feature`
- ---Forking Example
  - `git clone git@github.com:shyamkharwadkar/gitdem02.git`
  - `git remote add upstream`  
`https://github.com/maheshkharwadkar/gitdem02.git`
  - `git remote show`





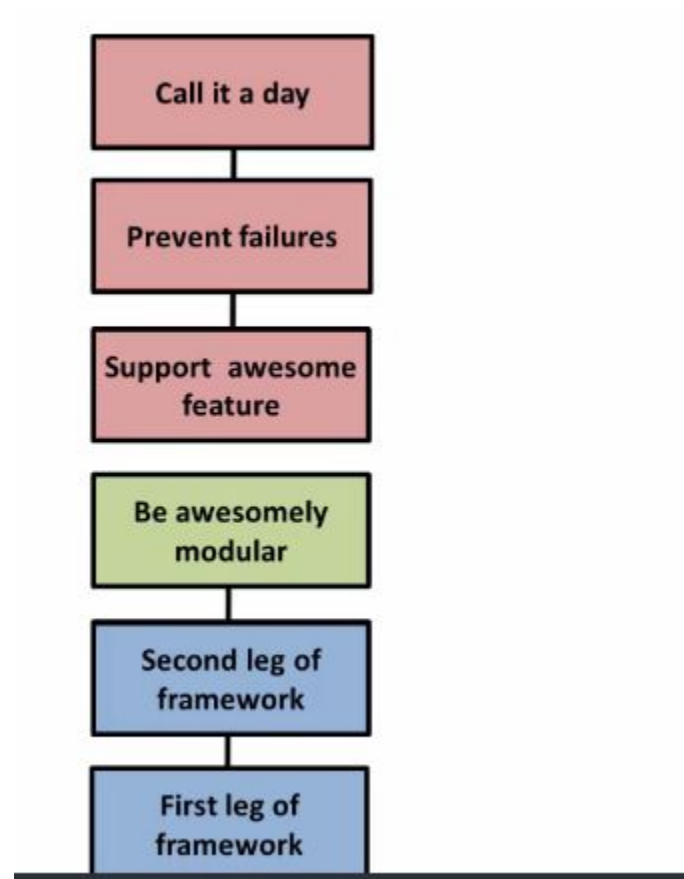


- `git fetch upstream`
- `git checkout --track upstream/my-awssome-feature`
- `git pull << Update local branch>>`
- `git fetch`
- `git status`



- Rebasing - Detaching your commits from the point of divergence
- **git pull --rebase**





- What is maven plugin
- What is maven plugin goal
- What is maven lifecycle
- What is maven lifecycle phase
- What are maven project coordinates
- What is maven project object model (POM)

`$mvn archetype:generate`

`$mvn pugin_Identifier:goal_Identifier`

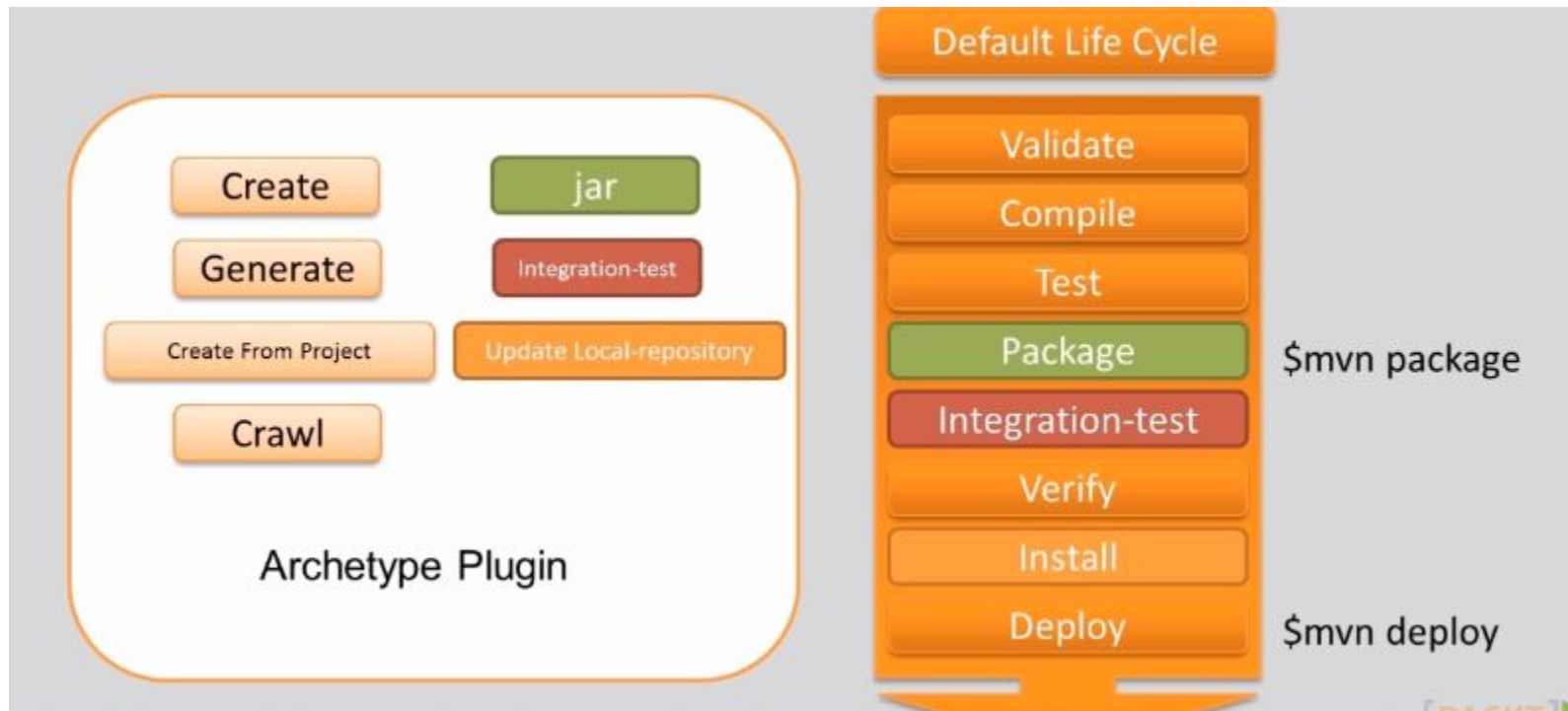
Goal 1

Goal 2

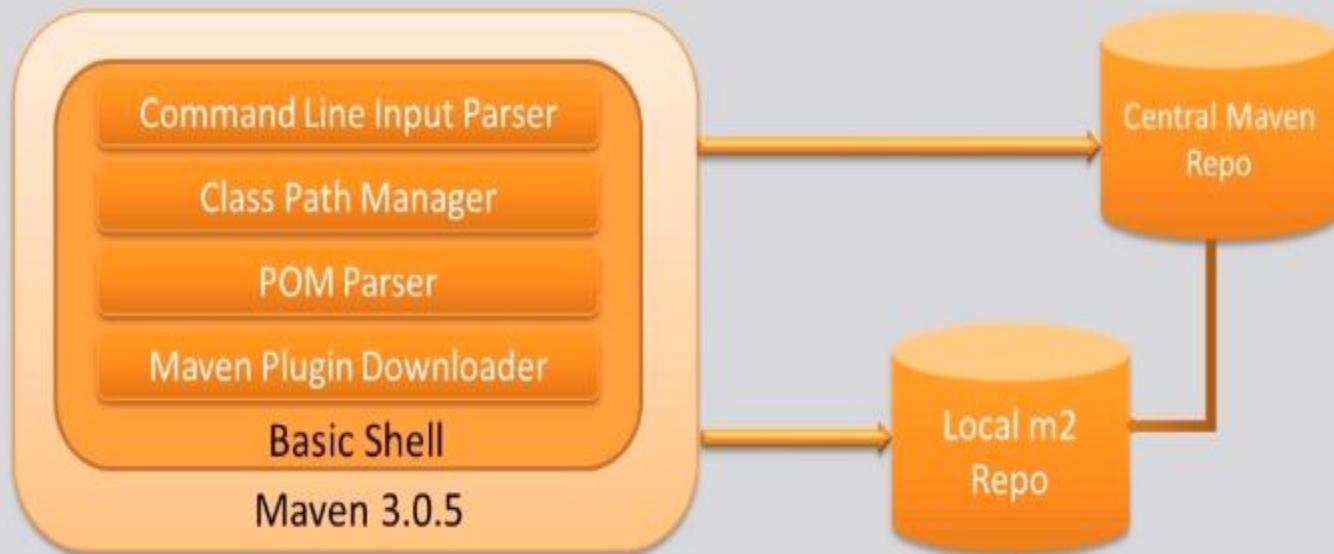
Goal 3

Plugin

# What is Maven Plugin



- All of its work is carried out by maven through its plugins
- Maven is a plugin execution framework



- Maven plugins can be executed in two ways
  - Direct execution
  - Lifecycle execution
- Built-in lifecycles of maven
  - Default
  - Clean
  - Site
- When we execute Default lifecycle
  - Validate the project
  - Compile the project sources
  - Run the project unit tests
  - Package the project binaries
  - Run integration tests against your project's package
  - Install the package into local repository
  - Finally deploy the package into the specified environment

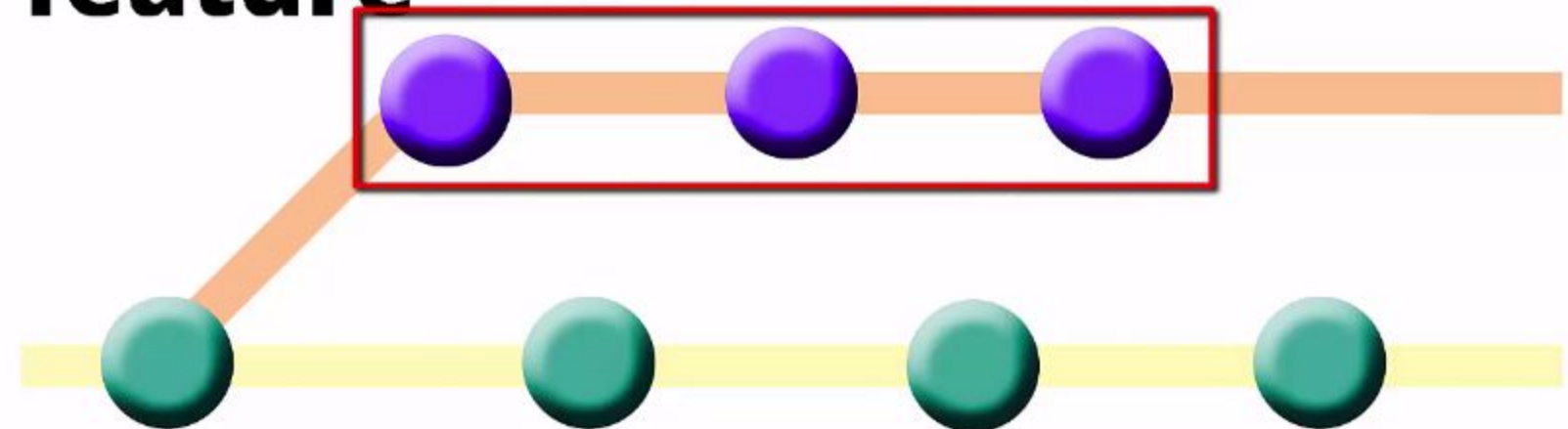
Build Phases	Goal
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResources
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar
install	install:install

## Simple Rebase Example

# Branching

**feature**

**Timeline**



**master**



## Simple Rebase Example

# Rebase

