FLEXMIND

GIT BASICS FOR EVERYONE

commit the code

- Sanjeev Jaiswal

FLEXMIND

Agenda

- What is VCS, History of VCS
- Meet git, Why git, git features
- Introduction to github
- Setting up GitHub and git for ssh access
- Minimal git commands

- `git config`
- `git init`
- `git clone`
- `git add`
- `git checkout`
- `git commit`
- `git pull/push`
- `git log`
- `git status`
- `git diff`
- `git branch`
- `git merge`
- `git reset`
- `git rm`
- `git stash`
- `git remote`

Course is for

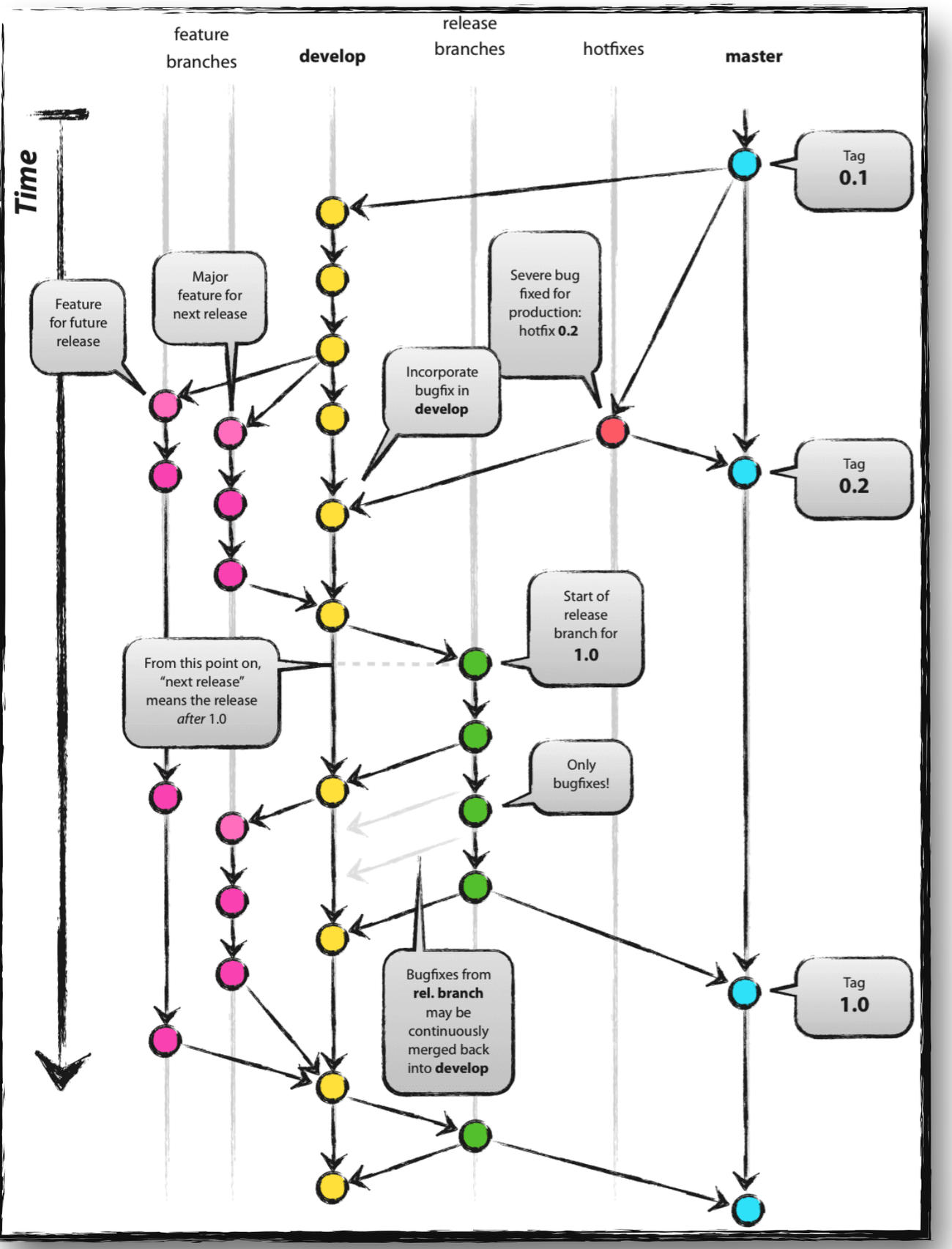
- Who wants to keep code safe remotely
- Anyone who wants to learn git commands
- Students/Freshers
- Developer
- Tester
- DevOps
- Hackers

Prerequisites for this course

- Worked on terminal before (Windows or Linux)
- Basics of *nix or Windows cmd commands
- Basic understanding of Software Application know-how

VERSION CONTROL SOFTWARE

don’t lose your hard work



Let's meet VCS

- What is Version Control Software
- Why we need it
- How it works
- Client-server vs Distributed
- Some known VCS

What and Why we need Version Control Software

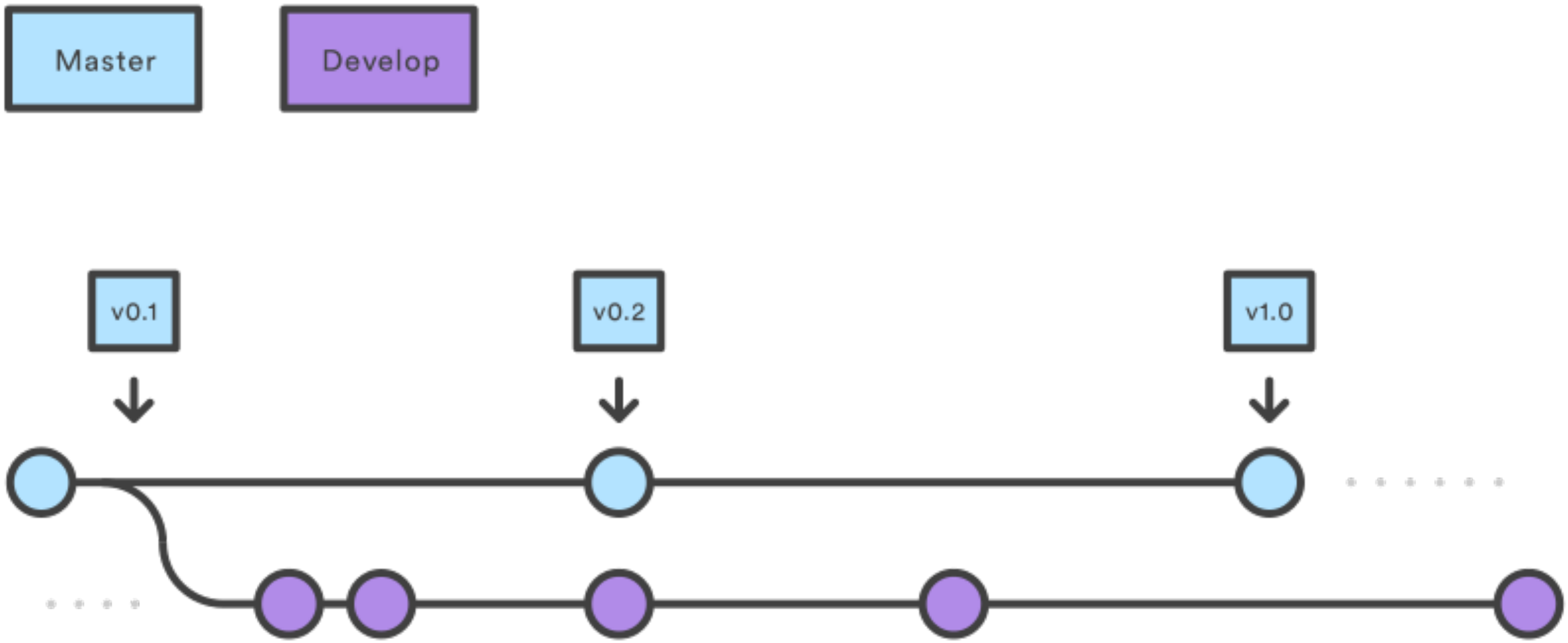
- SCM component
- To track every changes
- Maintain different versioning for Dev, QA, Prod
- Easy for collaboration
- Helps you develop and ship products faster
- It helps DevOps specially

How it works

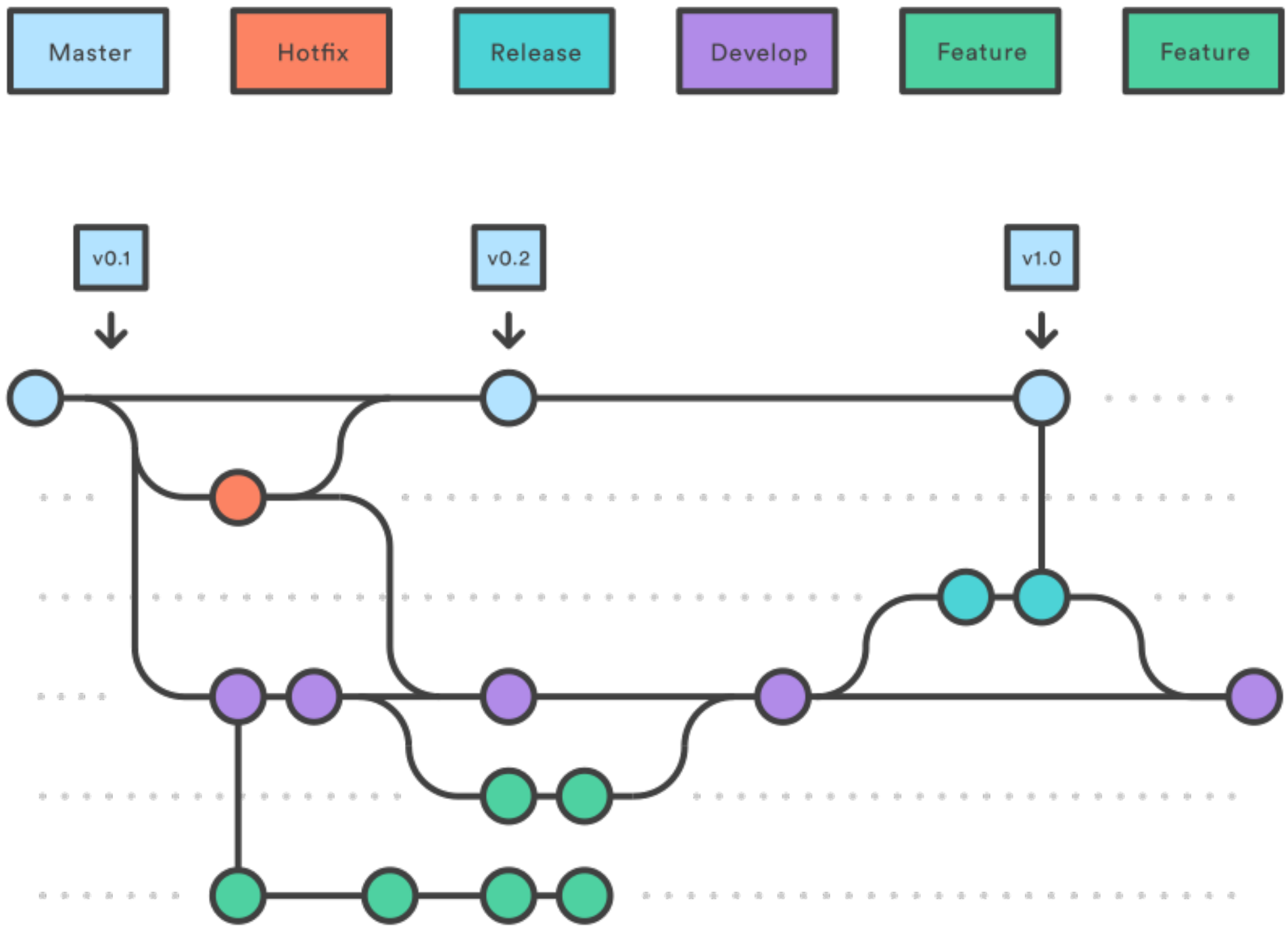
```
jassi@jazzmac  ~  git flow init
Initialized empty Git repository in /Users/jassi/.git/
No branches exist yet. Base branches must be created now.
Branch name for production releases: [master] prod
Branch name for "next release" development: [develop] release

How to name your supporting branch prefixes?
Feature branches? [feature/] feature
Release branches? [release/] release
Hotfix branches? [hotfix/] hotfix
Support branches? [support/] support
Version tag prefix? []
```

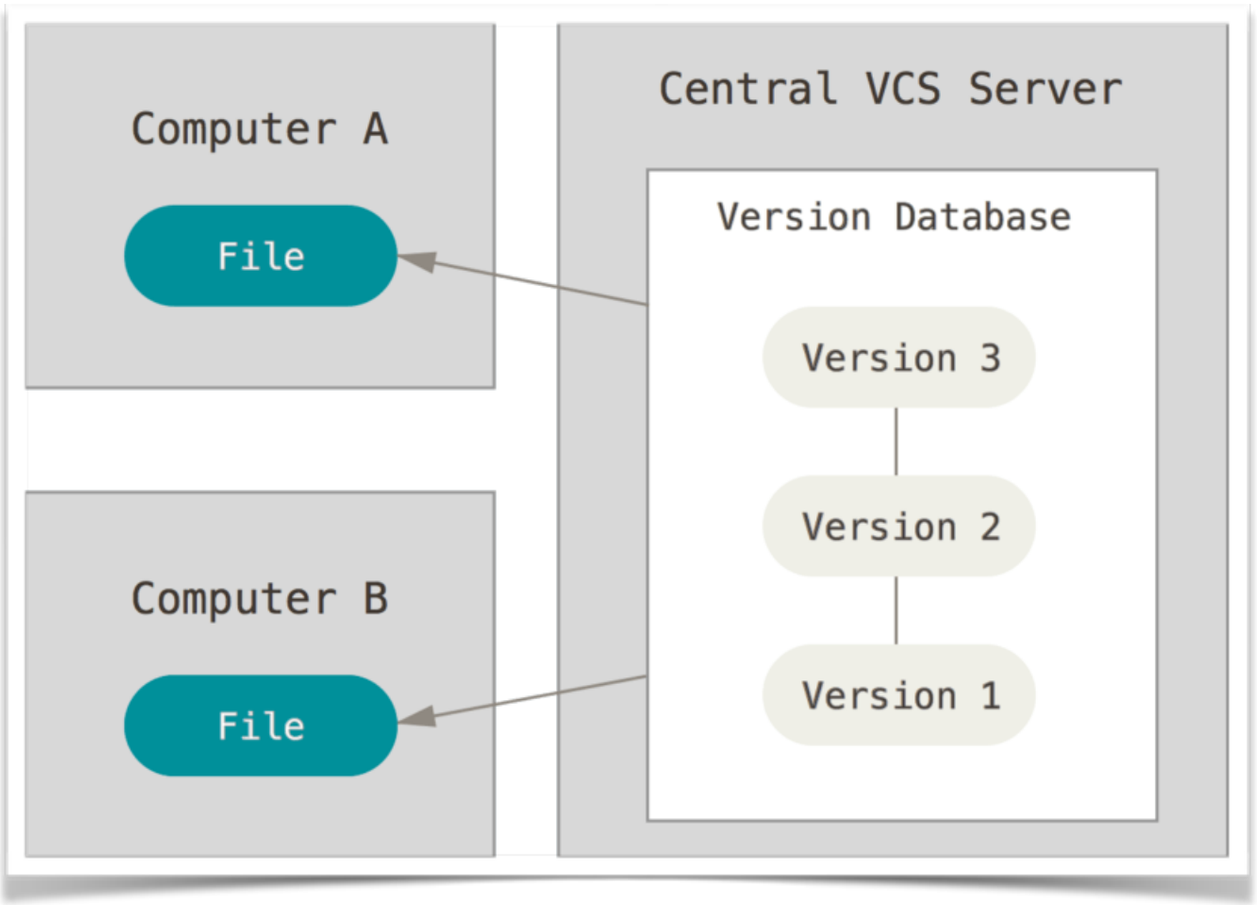
How it works continued

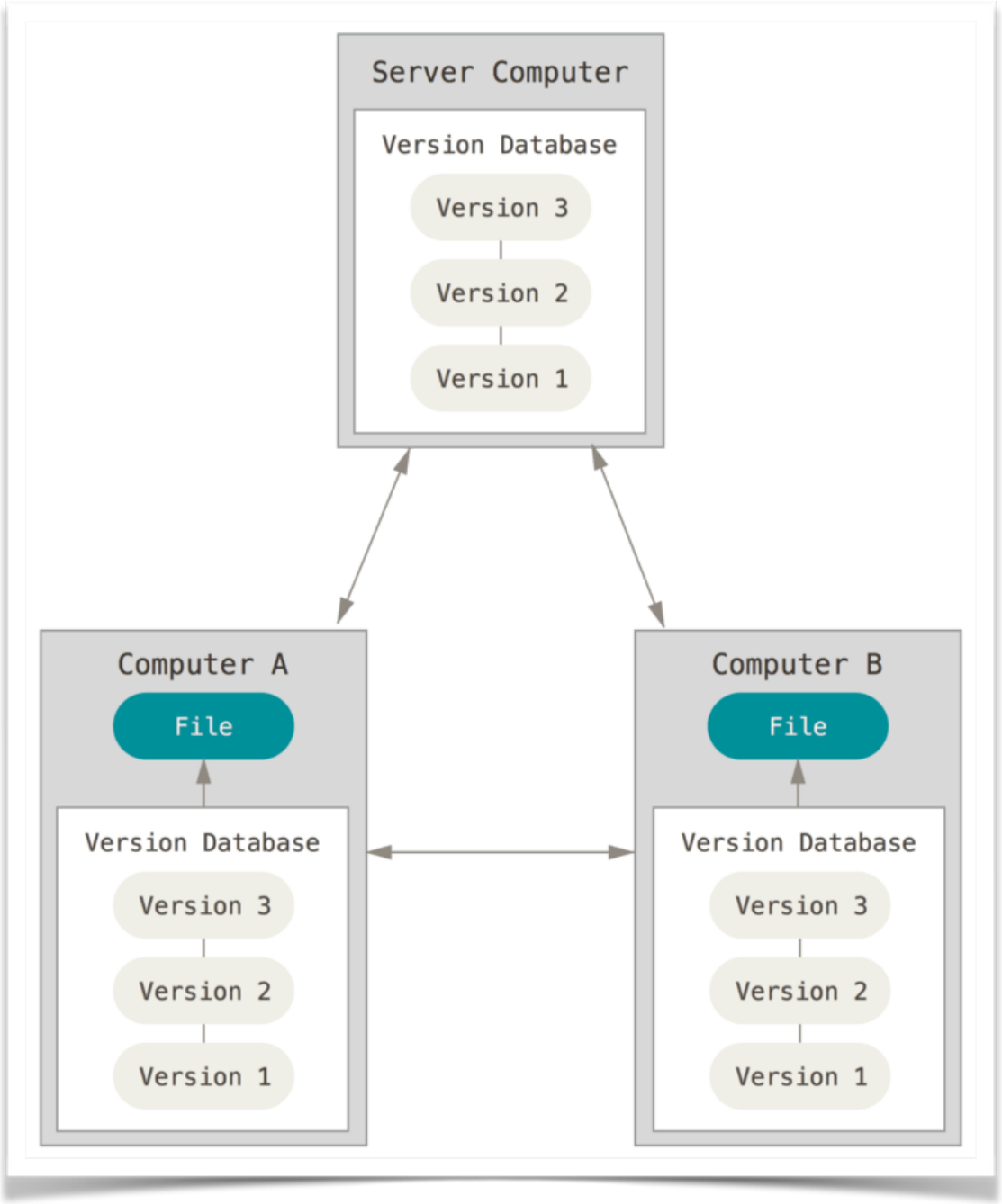


How it works continued



Client-Server (CVCS) vs Distributed VCS (DVCS)

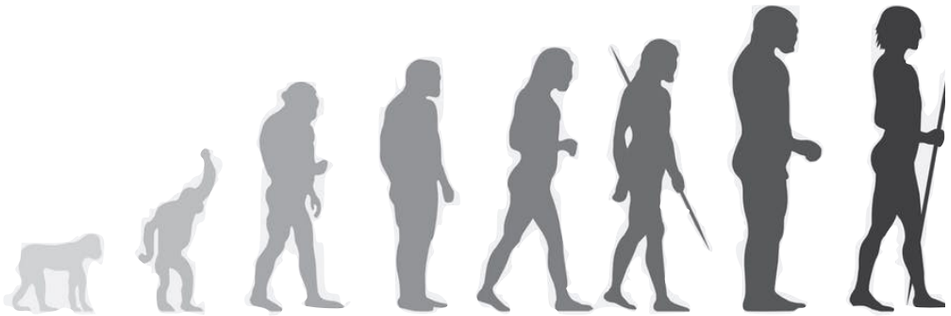




Learn more about it: https://nvie.com/posts/a-successful-git-branching-model/

SOME KNOWN VCS

Naming a few popular one



CVS (Concurrent Version systems)

- 1990s
- Written in C
- Linux
- Was part of GNU project
- Last stable release was in 2008
- http://savannah.nongnu.org/projects/cvs

Apache subversion

- commonly known as svn
- created by Collabnet in 2000, now an Apache project
- Written in C
- Compatible successor of CVS
- latest version: October, 2019
- Accenture, LinkedIn, Atmel, Codrus etc. uses it

Mercurial

- created in 2005
- written mainly in Python, but few C and Rust as well
- Lastest version: April, 2020
- Mozilla, nginx, OpenJDK
- Website: https://www.mercurial-scm.org/

Perforce

- perforce (Enterprise)
- Amazon used it in early days
- Google uses piper, based on perforce for private repo
- NetApp, TCS, Akamai, Amazon

There are many more

- GNU Bazaar
- BitKeeper
- Rational Clearcase
- Darcs
- Monotone
- Azure TFVC (Team Foundation Version Control)
- and so on …

MEET OUR HERO: GIT

makes developers life easier

flexmind.co

Hi Git, tell me something about yourself

- I was Created by Linux Torvalds in 2005
- Linux kept this name which means *unpleasant person* in British Language slang
- The man page describes Git as “the stupid content tracker”
- I am a Distributed Version Control system
- Written in collection of Perl, C and shell scripts
- Google, Quora, Facebook, netflix, reddit, lyft etc. utilise me
- Latest version (2.27): June, 2020
- Visit me at: https://git-scm.com/

Git installation

- Macbook:
  - If Xcode exists, probably you already have git.
    Check with `git —version`
  - `brew install git`
- Windows:
  - Download windows git installer
  - Install it with default options
- Linux:
  - `sudo yum install git`
  - `sudo apt-get install git`
- Here is the installation guide: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

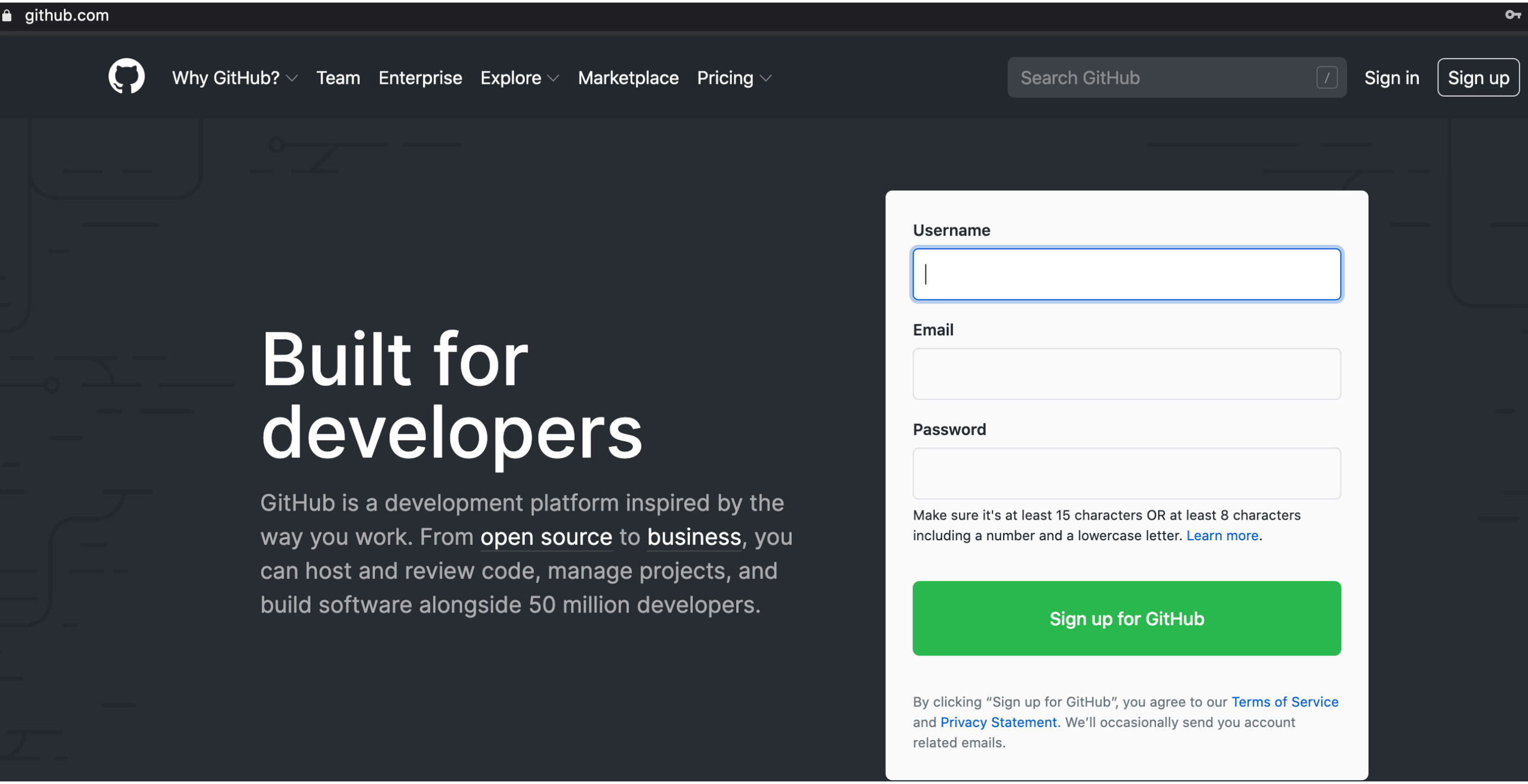
GITHUB

git with GitHub, a deadly combo

Minimal introduction to GitHub

- Online Code repository
- Bug Tracking
- Branches
- Team Management
- Project Management
- Workflow automation
- Helps in secure development
- Better Code Review
- Read More here: https://github.com/features

GITHUB ACCOUNT SETUP AND WALKTHROUGH


github.com
Why GitHub?
Team
Enterprise
Explore
Marketplace
Pricing
Search GitHub
/
Sign in
Sign up
Built for developers
GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside 50 million developers.
Username
Email
Password
Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.
Sign up for GitHub
By clicking "Sign up for GitHub", you agree to our Terms of Service and Privacy Statement. We'll occasionally send you account related emails.

WALKTHROUGH OF YOUR PROFILE

WALKTHROUGH OF YOUR REPOSITORY

fork, PR, Bug Track etc.

- Watch, Star
- Clone, Fork
- Pull Request
- Wiki
- Marketplace
- Explore, Trending
- Search Repo

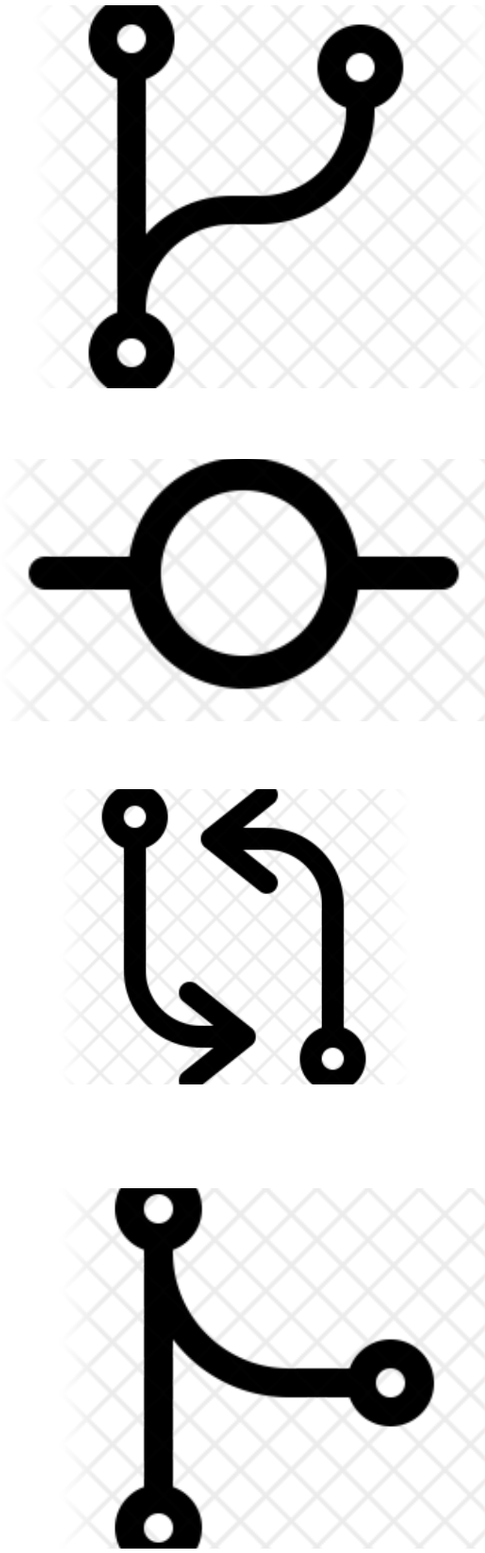
ssh setup for github

- Check if key already exists
  - `ls -la ~/.ssh and look for id_xxx.pub`
- Generate a new ssh key
  - `ssh-keygen -t rsa -b 4096 -C “your_email@example.com”`
- Add ssh key to ssh-agent
  - `eval “$(ssh-agent -s)”`
  - Check if config file exists ~/.ssh/config, if not create a new one
  - Add below lines

```
Host *
  AddKeysToAgent yes
  UseKeychain yes
  IdentityFile ~/.ssh/id_rsa
```

- Add private key: ssh-add -K ~/.ssh/id_rsa
- Add your newly created ssh public key to your GitHub account

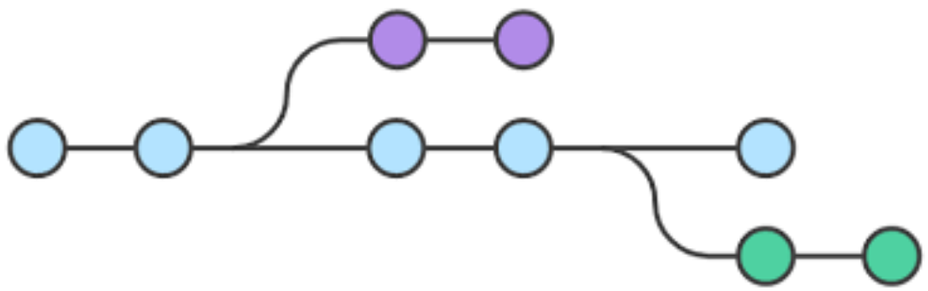
GIT COMMANDS



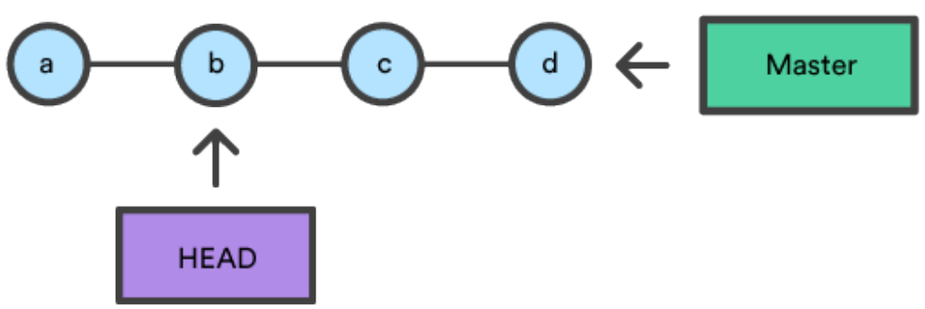
minimal commands to learn

What's branch, HEAD, master, commit

Branch



HEAD



Commit flow

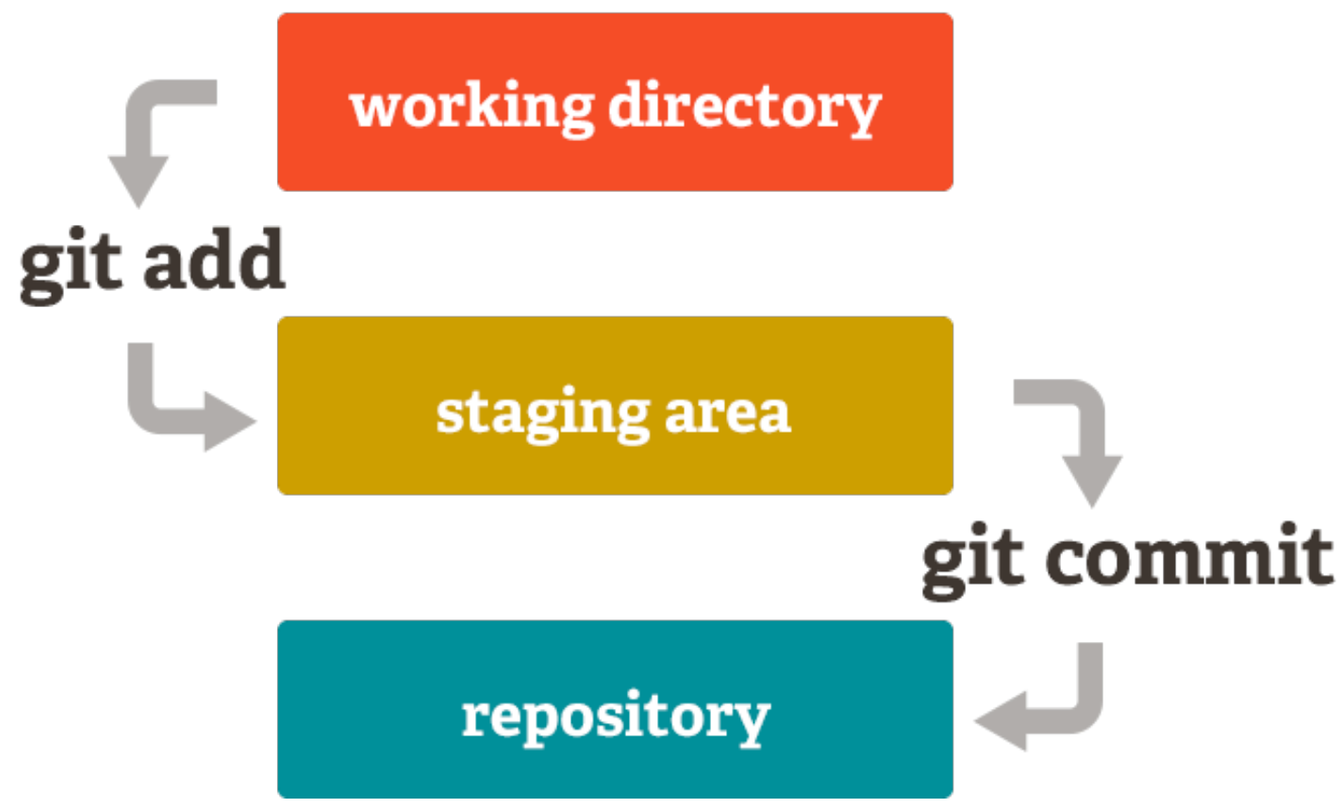


Image source: atlassian and medium

Common git commands everyone should know

- git config
- git init
- git clone
- git remote
- git add
- git commit
- git status
- git diff
- git log
- git pull
- git push
- git checkout
- git branch
- git tag
- git reset
- git merge
- git rm
- git stash

SETTING UP REPOSITORY

git init

- Initialises empty git repo(sitory)
- Creates .git folder
- **.git folder** contains all the information necessary for your project in version control
- Also contains the information about commits, remote repository address, etc.
- It also contains a log that stores your commit history

```
staff    25B Jul 11 12:31 COMMIT_EDITMSG
staff    23B Jul 11 12:31 HEAD
staff   137B Jul 11 12:22 config
staff    73B Jul 11 12:22 description
staff   448B Jul 11 12:22 hooks
staff   137B Jul 11 12:31 index
staff    96B Jul 11 12:22 info
staff   128B Jul 11 12:29 logs
staff   320B Jul 11 12:31 objects
staff   128B Jul 11 12:22 refs
```

git config

- Setup environment config
- You can configure globally or locally
- git config
- Common commands:
  - git config --list
  - git config --get user.name
  - git config --add ; add new variable: name value
  - git config --local|global user.name “user-name”
  - git config --local|global user.email “user-email”

.gitignore

- Ignore files/folders that you don't want to commit
- It can be local and global as well
- Create .gitignore file inside a root directory of the repo
- git config --global core.excludesfile ~/.gitignore_global
- Some sample lines for .gitignore

```
# Ignore Mac system files
.DS_store

# Ignore node_modules folder
node_modules

# Ignore all text files
*.txt
*.csv

# Ignore environment file
.env or some .ini
```

- .gitignore templates from github: https://github.com/github/gitignore

Summary

What we learned so far

- Setting up the empty repo: git init
- Configure name, email at least: git config
- Ignore files/folders that you want to ignore: .gitignore
- How to create alias of some frequent git commands

Lab – setting up repository

Time: 20 minutes

1. Set up the empty repo using git init command
2. Configure name, email at least using git config command
3. Ignore files/folders that you want to ignore
    1. create .gitignore using your favourite editor (vim|nano)
    2. Add few extensions or file name that you want to ignore
    3. Add files/directories to be ignored and save .gitignore file
4. Check with git status command if it's really ignored

SAVING CHANGES

git add

- Adds the changes in staging area to track files for commit
- git add .
- git add *
- git add <file-name>
- git add sub-dir/*.txt
- git add lab-*.txt # wont add sub-dir/lab-3a.txt
- git add -n # dry run
- git add -i #interactive mode

git commit

- Commit the staged/tracked files
- git commit
- git commit -m “commit message”
- git commit -am “commit message”
- git commit --amend # forgot something to commit in past?
- git commit --dry-run

git status

- Displays the status of working directory and the staging area
- git status # mostly used command and meaningful
- git status --long # by default
- git status -s # short; one line
- git status -b #branch status
- git status --show-stash #stashed files
- git status --help

git diff

- Differentiate the content with last commit
- git diff # changes since last commit
- git diff <file-name>
- git diff HEAD <file-name>
- git diff HEAD^ HEAD
- git diff --cached <file-name> #staged changes with the local repository
- git diff master other-branch <file-name>

git stash

- You are not ready to commit but wants to work on something else. Well, git stash comes handy here.
- git stash #saves uncommitted changes for later use
- git stash -u #untracked
- git stash -a # everything including ignore files
- git stash pop
- git stash apply n
- git stash list
- git stash show
- Note: git stash needs at least one initial commit to work

git log

- View the commit history
- git log
- git log -n # n is integer
- git log --oneline
- git log --pretty=oneline
- git log --oneline -n
- git log --oneline --graph --all
- git log -p -2
- git log --stat [-n]
- Check git log filters (author, range, date, string, file etc.)

git show

- Examine specific revisions
- git show <first-4char-commit-id>
- git show [HEAD | HEAD~ | HEAD~1]
- git show master~3
- git show <7charHash>~~~
- git show master^
- git show HEAD^2 #2nd parent of a merge commit
- git show HEAD^^

Summary

- Added/modified files
- Checked the status
- Committed staged files
- Learned why and when to stash files
- Checked difference in edited files
- Inspected various commit logs

Lab – Saving changes

Time: 45 minutes

1. See the current status of repo using git status command
2. Create few files and add some contents in those files
3. Add those files in staging area using git add command
4. Check the status and try to understand the output
5. Commit those files using git commit commands
6. Edit some contents of some files
7. Check the difference using git diff command
8. Practice git stash command
9. Check the status again
10. Add and commit again
11. Check the log using git log command

MERGING & BRANCHING

git branch/checkout

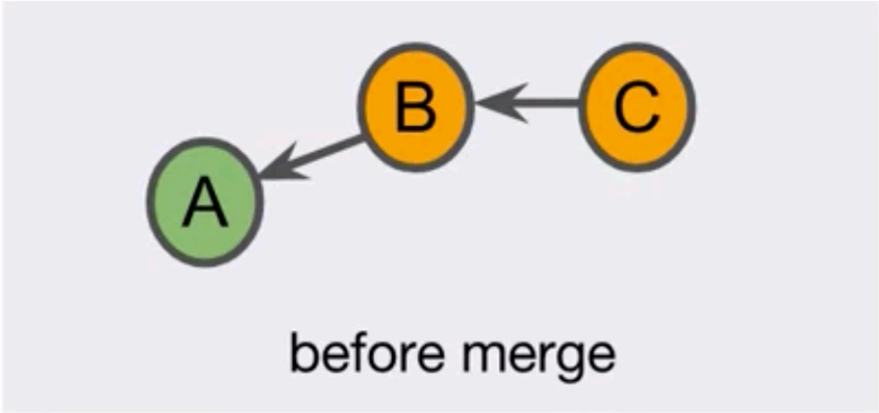
- Branch is another label to work on something separately
- git branch or git branch -l|-a or git show-branch (more details)
- git branch <branch-name>
- git branch -d <branch-name>
- git checkout <branch-name>
- git checkout <commit-hash> # Detached HEAD
- git checkout -- master # guess what it does? Confusing?
- git checkout -b branch-name
- git branch -D <branch-name> # Dangling commits
- git reflog # find dangling commit
- git checkout -b <branch-name> <7-digit-dangling-commit>
- Home work: How to checkout remote branch?

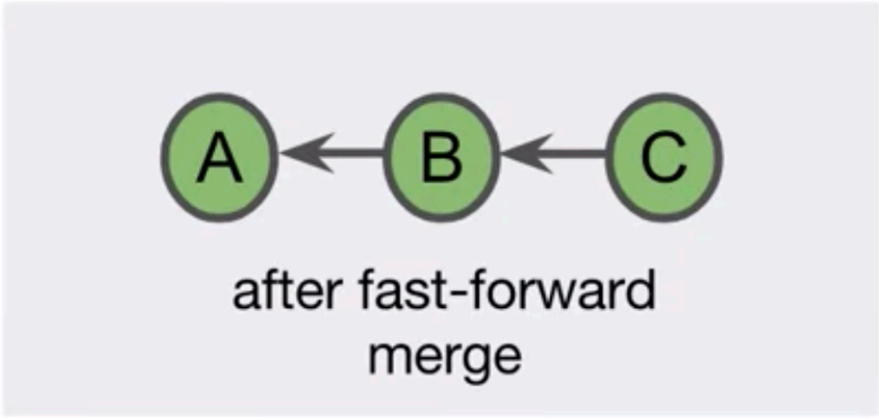
git switch

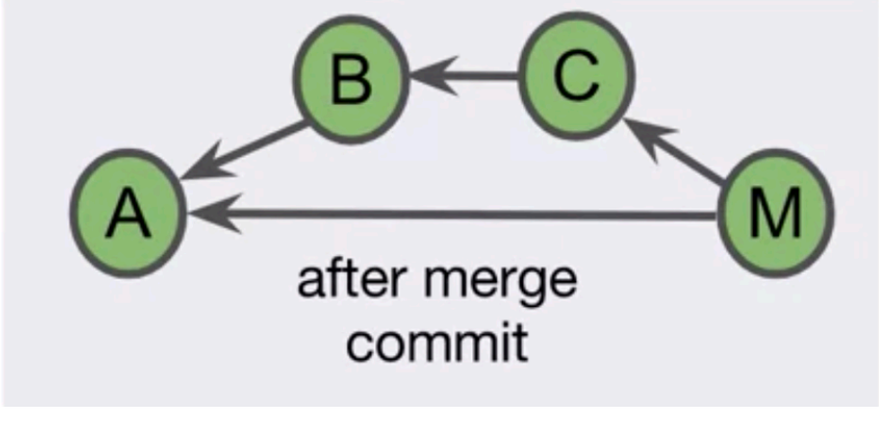
- Switches to another branch (from git-2.23 : Aug, 2019)
- git switch branch-name
- git switch -c branch-name
- git switch -
- git switch -c backfix HEAD~2
- git switch --detach HEAD~3
- git switch -c keepit

git merge

- Fast-forward merge
  - By default
  - git <merge branch-name>
  - Linear commit graph
  - Not possible if master branch also has some commits
- Merge commit (3-way-merge)
  - non-linear commit graph
  - policy to follow merge-commit always rather than fast-forward
  - git merge --no-ff <branch-name>







Dealing with merge conflicts

- When you will get merge conflicts?
  - When two branches change the same file, hard to decide
  - git modifies file(s) and places them in working tree
- When you won't get merge conflicts?
  - changed files are different
  - Even it's same file, but easy to decide for git
- How you can avoid merge conflicts?
  - Apply pull-merge-commit-push wherever possible
  - Merge commits to master branch more often

Dealing with merge conflicts

How you resolve merge conflicts?

1. Checkout master
2. Merge branchX
    1. got a conflict in fileA
3. Fix fileA (Here you need manual checks)
4. Perform merge commit
5. Delete the branchX label

git tag

- Tag is a reference to a specific commit
- Lightweight and Annotated tag
- git tag
- git tag tagname
- git tag -a tagname -m “message”
- git tag <tagname> <commit> #defaults to HEAD
- git tag v0.1 HEAD~3 # tag the previous commit
- git tag -a -m “includes security fix” v1.3
- git show v1.3
- git push <remote> <tagname> #single tag
- git push <remote> --tags # all tags

Summary

- Use of branch using git branch and git checkout command
- git checkout alternative as git switch command
- Why would you need to merge some commits
- git merge command usage
- Learned various way of avoiding and merging the conflicts
- Used git tag command as well
- How to delete branch
- What is detached HEAD and dangling commits

Lab - Merging & Branching

Time: 30 minutes

1. Create a new branch
    1. using git branch
    2. also using git checkout
2. Switch into that branch
    1. using git checkout
    2. using git switch
3. Merge the conflicts
    1. Fast forward
    2. merge commit
4. Use tag for some commits
5. Delete the branch
6. Also practice for detached HEAD and Dangling commits (Optional)

SYNCING REPOSITORIES

git clone

- Clone the copy of a target repo
- Can be local or remote (mostly)
- Different cloning scenarios
  - You have local repo, want to merge with remote repo
    - existing remote repo
    - newly created remote repo
  - Clone remote repo

git clone

- git clone <remote-repo-url> <local-repo-name>
- git clone -branch <branch-name> <remote-repo>
- Can be cloned using https or ssh protocol (recommended)
  - git@github.com:jassics/awesome-aws-security.git
  - https://github.com/jassics/awesome-aws-security.git
- clone vs fork

git remote

- Working with remote repo
- git remote
- git remote -v
- Check the contents of .git/config file
- git remote show <name>
- git remote add <name> <url>
- git remote set-url <name> <url>
- git remote rename <old-name> <new-name>
- git remote rm <name>

git fetch

- Fetches new objects and references from the remote repo
- Good way to know what others are doing on that repo
- Git isolates fetched content from an existing local repo
- git fetch <remote>
- git fetch <remote> <branch>
- git fetch --all
- git fetch --dry-run
- git fetch --prune

git pull

- Fetches and merges commit locally/ Pull out the updates
- git pull <remote-branch> <local-branch>
  - git pull origin/master master or
  - git pull origin dev
- Combination of git fetch and git merge

git push

- Pushes new objects and reference to the remote repo
- git push <remote> <branch>
  - Ex: git push origin master
- git push <remote> --all # push all branches
- git push <remote> --tags # push all tags to remote repo
- git push origin local_branch:remote_branch #push branch
- git push origin :branch_name #delete branch remotely

Summary

- Clone the repository using git clone
- Usage of git remote command
- How git fetch works
- What git pull does for you
- How you pushed the changes using git push command

Lab - Syncing Repositories

Time: 45 minutes

1. Clone the repository from github (through ssh) using git clone command
2. Check if remote repo url is ok using git remote command
3. Do some changes in files
4. Add and commit. Be ready to push
5. Check if something got committed remotely by using git pull command
6. Merge if there is any conflict
7. Push the changes using git push command

Course Summary

- What is Version Control Software (VCS)
- Walkthrough of git and github features
- Essential commands
  - setting up repo
  - saving changes
  - merging and branching
  - syncing repositories
- Labs for essential commands
- What's Next

What's Next

- Git Advanced commands (submodules, blame, revert, rebase)
- Pull Request
- Types of Git workflow
- Github Administration
- Github Actions
- Webhook Documentation
- Try to use IDE like PyCharm for git GUI option

Learning Resources

- Pro git Book (Free to read)
- Learn by Doing from github
- Learn Git branching
- Learn form Git Tower
- Github Lab
- Learn git from Atlassian (Remember JIRA, Bitbucket, Confluence)
- Version Control Software (Wiki)

Twitter: twitter.com/flexmind_co

***Linkedin**: linkedin.com/in/flexmind*

***Website:** https://flexmind.co/*

***Contact Us:** learning@flexmind.co*

Follow me in Twitter: @jassics

THANK YOU GUYS.
LIKE, SUBSCRIBE AND SHARE