GIT 101

glua

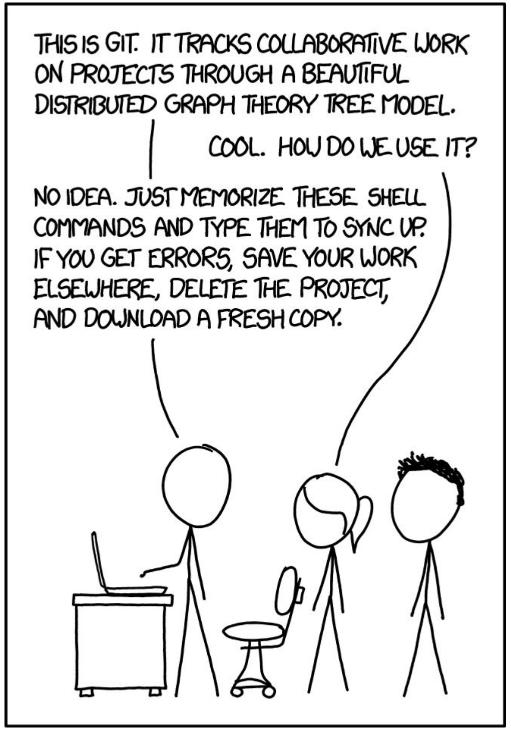
universidade de aveiro

instituto de telecomunicações

Version Control

- ❑ Lots of people working on the same code
- ❑ Stores your code: history of **Who** did **What** and **When**!
- ❑ Fine grained control of changes
  - ❑ Messed up? revert
  - ❑ Merge code from different people
- ❑ Prevents heart attacks. Developer safety net.

Return to Zero
WHAT'S THIS CUP OF USB DRIVES?
AH, THAT'S HOW I VERSION-CONTROL MY FILES...
TYPE
TYPE TYPE
STAN'S DESK!
IT'S BRILLANT REALLY. JUST MAKE A NEW COPY ON A NEW ONE EVERYDAY.
WHAT HAPPENS WHEN YOUR CUP FILLS UP?
EASY! I DUMP THEM IN HERE!
USB
EEWeb.com


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

GIT HISTORY

1. In 2005 Linus Torvalds needed a version control system for the kernel - fast and safe

2. Someone else wanted **git** to support a workflow for [RANDOM WORKFLOW HERE]

3. Repeat step 2 (for >10 years)

Terminology

- ❏ **Commit**: is a change in one or more files, with a helpful message
- ❏ **Branch**: A sequence of commits. Usually each branch matches a flow of work (e.g. bugfix, or new feature)
- ❏ **Remote**: Remote git server
- ❏ Convention: master is the main development branch and origin is the default server where you push/fetch

Setup

- ❏ `$ sudo apt install git-all (debian)`
- ❏ `$ sudo dnf install git-all (redhat)`

- ❏ `$ git --version (should work)`

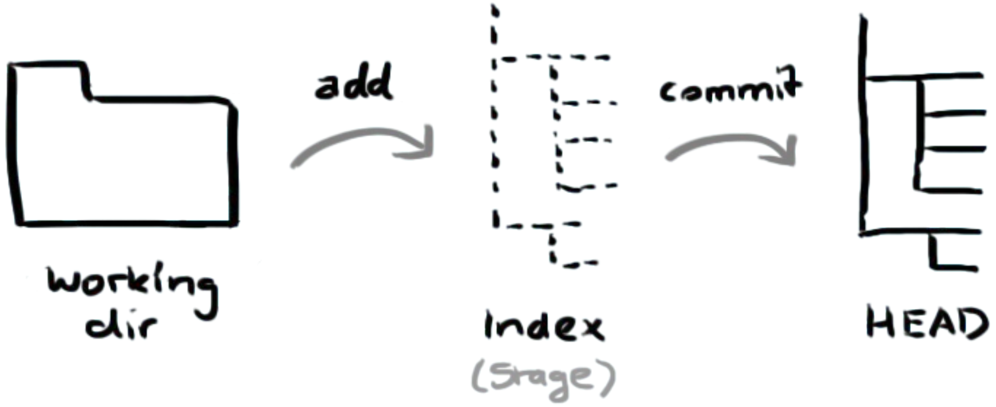
CREATE/CHECKOUT A NEW REPOSITORY

- Create a **local** repository
  - `$git init`

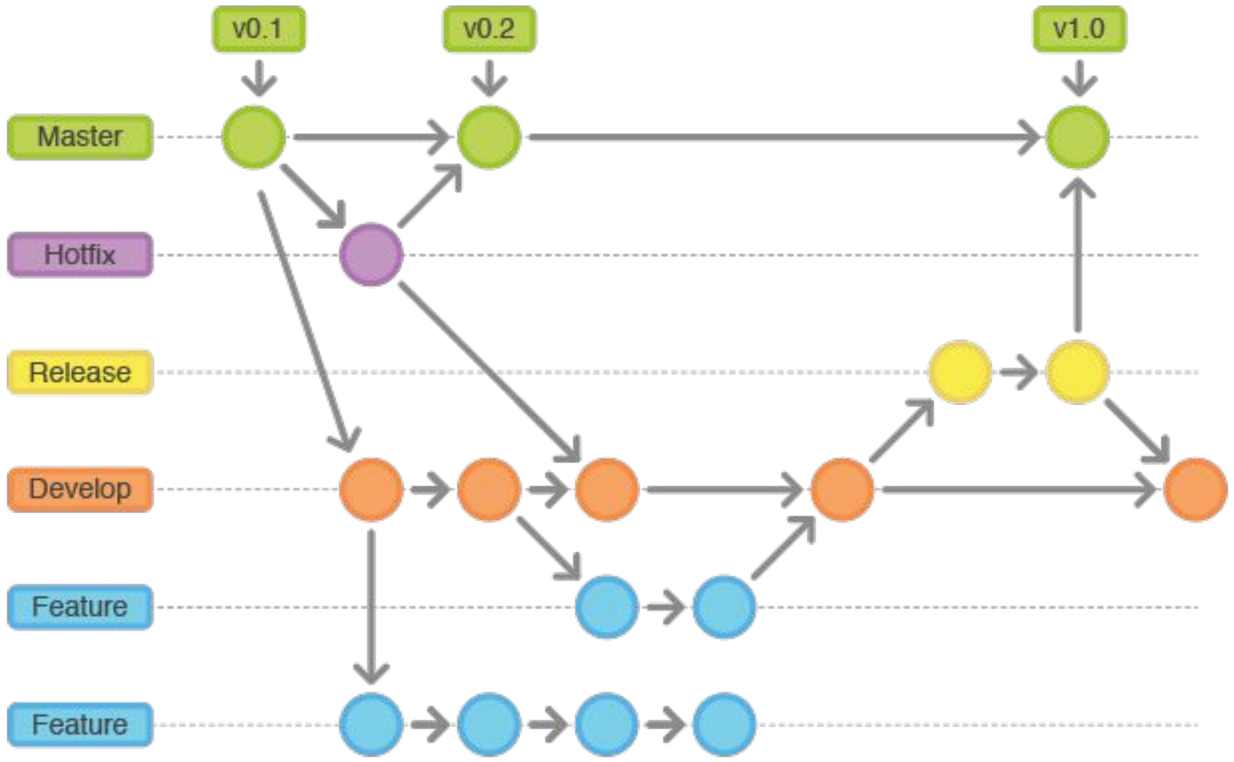
- Checkout a **remote** repository
  - `$git clone /path/to/repository`
  - `$git clone username@host:/path/to/repository`

Workflow

- Local repository consists of three "trees":
  - **Working Directory** (actual files)
  - **Index** which acts as a staging area
  - **HEAD** which points to the last commit



Workflow



LOG

- Study repository history
  - `$git log`
  - `$git log --author=bob`
  - `$git log --pretty=oneline`
  - `$git log --graph --oneline --decorate --all`
  - `$git log --name-status`
  - `git log --help`

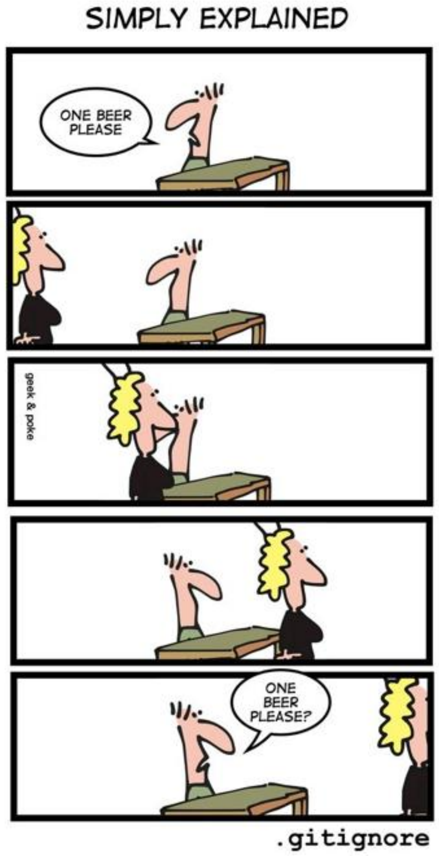
ADD & COMMIT

- ❑ Add files to the staging area (**Index**)
  - ❑ `$git add <filename>`
  - ❑ `$git add * (so not forget .gitignore)`
- ❑ Commit the current version (**Head**)
  - ❑ `$git commit -m "Commit message"`
  - ❑ `$git commit -a -m "Commit message" (avoids adding the files)`

- ❑ Check the status **Working Dir.**
  - ❑ `$git status`

.Gitignore

```
# hidden files
.*
# backup files
*.bak
# dont ignore .gitignore :D
!.gitignore
# Objects files
*.class
*.o
```



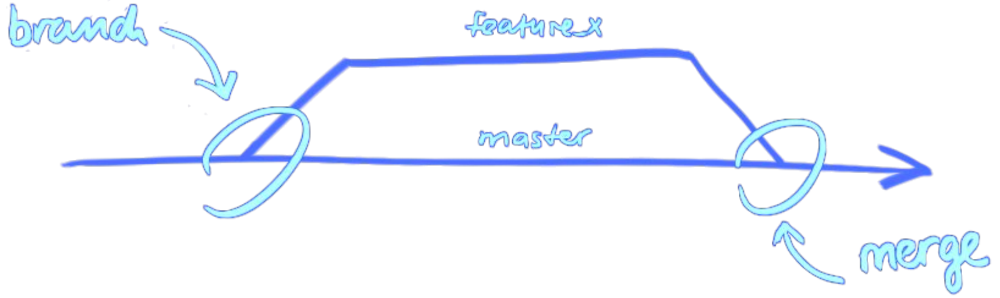
PUSHING CHANGES (REMOTE)

- Commit to a remote repository
  - $git push
  - $git push origin <master>

- Add remote server
  - git remote add origin <server>

BRANCHING

- Branches are used to develop code isolated
- Create a new branch
  - `$git checkout -b feature_x`
- Switch back to **master**
  - `$git checkout master`
- Delete a branch
  - `$git branch -d feature_x`
- A branch is not available to others
  - `$git push origin <branch>`



UPDATE & MERGE

- Update your local repository to the newest commit
  - $git pull
- Before merging
  - $git diff <source_branch> <target_branch>
- Merge another branch into your active branch
  - $git merge <branch> (auto merge)
- In case of conflicts
  - Apply manual resolution
  - $git add <filename>
  - $git commit -a -m "Commit message"

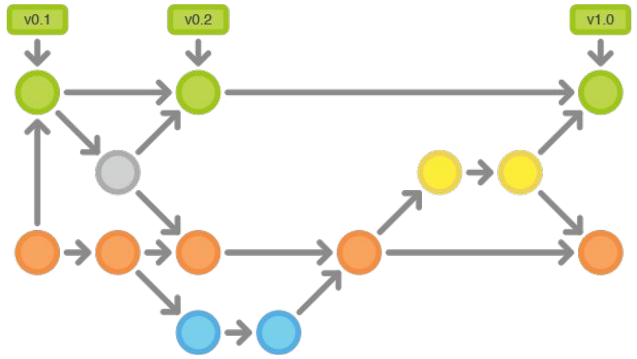
REPLACE LOCAL CHANGES

- In case you did something wrong (for sure never happens)
- Replace local changes
    - `$git checkout -- <filename>`

- Drop all your local changes and commits
    - `$git fetch origin`
    - `$git reset --hard origin/master`

TAGGING

- Recommended to create tags for software releases
    - `$git tag 1.0.0 1b2e1d63ff`

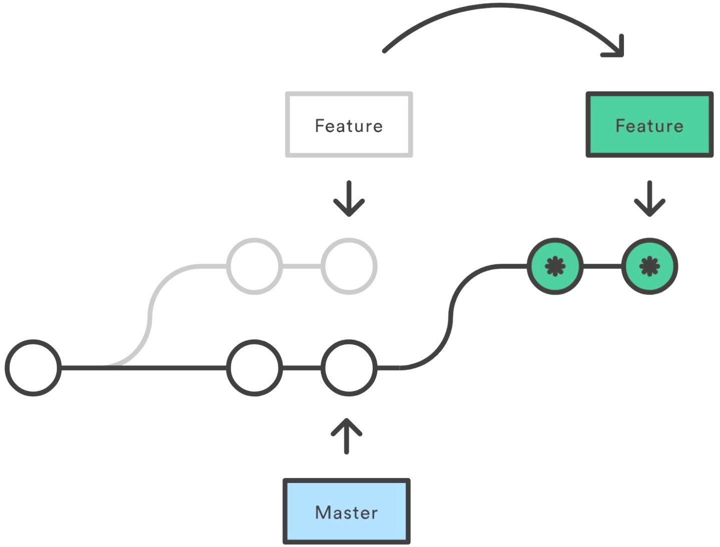


Git Upstream

- ❑ Sync changes from the original repository to your fork
- ❑ List the current configured remotes
  - ❑ `$ git remote -v`
- ❑ Specify a new remote upstream
  - ❑ `$ git remote add upstream` https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git
- ❑ Update your fork
  - ❑ `$ git fetch upstream`
  - ❑ `$ git checkout master`
  - ❑ `$ git merge upstream/master`

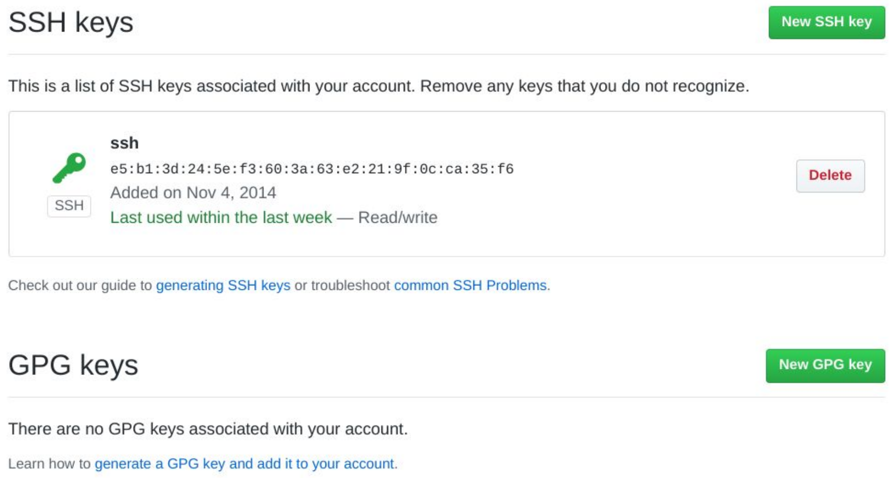
Git Rebase

- ❑ Another way to integrate changes from one branch to another
- ❑ Rebase compresses all the changes into a single “patch”
  - ❑ `$ git checkout feature`
  - ❑ `$ git rebase master`



GitHub Authentication

- Two ways to authenticate
  - HTTPS username and password
  - SSH Keys
- SSH Keys does not require you to enter a password
- Generate SSH key pair
  - $ ssh-keygen
- Add the content of your public key to GitHub
  - $ cat ~/.ssh/id_rsa.pub



SSH keys

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

ssh
`e5:b1:3d:24:5e:f3:60:3a:63:e2:21:9f:0c:ca:35:f6`
Added on Nov 4, 2014
Last used within the last week — Read/write

Check out our guide to generating SSH keys or troubleshoot common SSH Problems.

GPG keys

There are no GPG keys associated with your account.

Learn how to generate a GPG key and add it to your account.

References

- $git help
- https://www.git-scm.com/book/en/v2
- http://rogerdudler.github.io/git-guide
- https://github.com/equalsraf/git-talk/blob/gh-pages/git.md
- https://learngitbranching.js.org/
- http://stevelosh.com/blog/2013/04/git-koans/
- http://firstaidgit.io/#/

In case of fire

1. git commit
2. git push
3. leave building