Git: Introduction

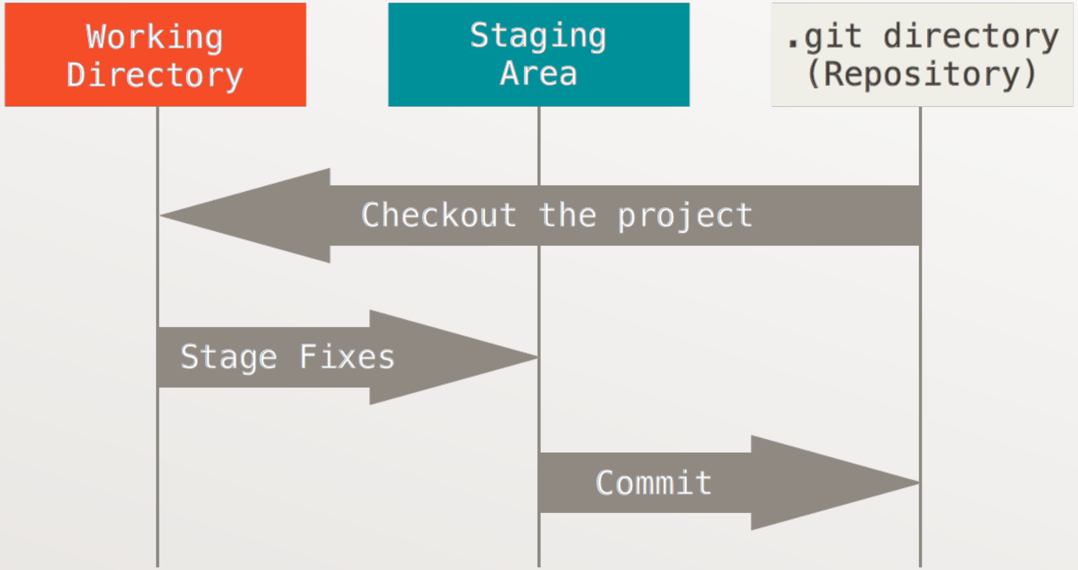
Content: Pato Lankenau

Graphics: git-scm.com, atlassian.com, rogerdudler.github.io

What is Git?

- Distributed version control tool (not file history)
- Made by Linus Torvalds

Staging


Working Directory
Staging Area
.git directory (Repository)
Checkout the project
Stage Fixes
Commit

Basic Commands

Creating local git repository

1. Create a folder and change into it
2. *git init*

Git and files

- Git keeps track of files and changes within them
- Files that git knows about are known as tracked files

Adding files to git index (stage)

git add <filename>

1. Create a file, helloworld.py
2. *git status*
3. *git add helloworld.py*
4. Edit file (*vim/nano/emacs/ed/gedit helloworld.py*)
5. Insert *print “Hello class”* or *print(“Hello class”)* for python3
6. *git status*

Committing changes

- When you change files, git knows
- *git status*
- *git commit <file>* or *git commit -a* or *git commit -m “commit message”*
- *git status*
- Clean

Important Git File: .gitignore

- Filenames that match any of these rules are ignored by git
- UNLESS they're already tracked
- Typical .gitignore

```
#Vim temp files
*.swp
#Compiled source
*.pyc
*.class
*.o
*.so
#OS generated files
.DS_Store
.DS_Store?
._*
.Spotlight-V100
.Trashes
```

Adding a gitignore file

1. Create a file named *.gitignore*
2. Add **.pyc*
3. Save
4. *git status*
5. *git add .gitignore*
6. *git commit -am “added gitignore”*
7. Compiled python files now ignored forever!

Making more changes

- Edit *helloworld.py*
- Change it to print *“hello world”*

- *git status*
- *git diff <commit> <filename>* or *git diff*

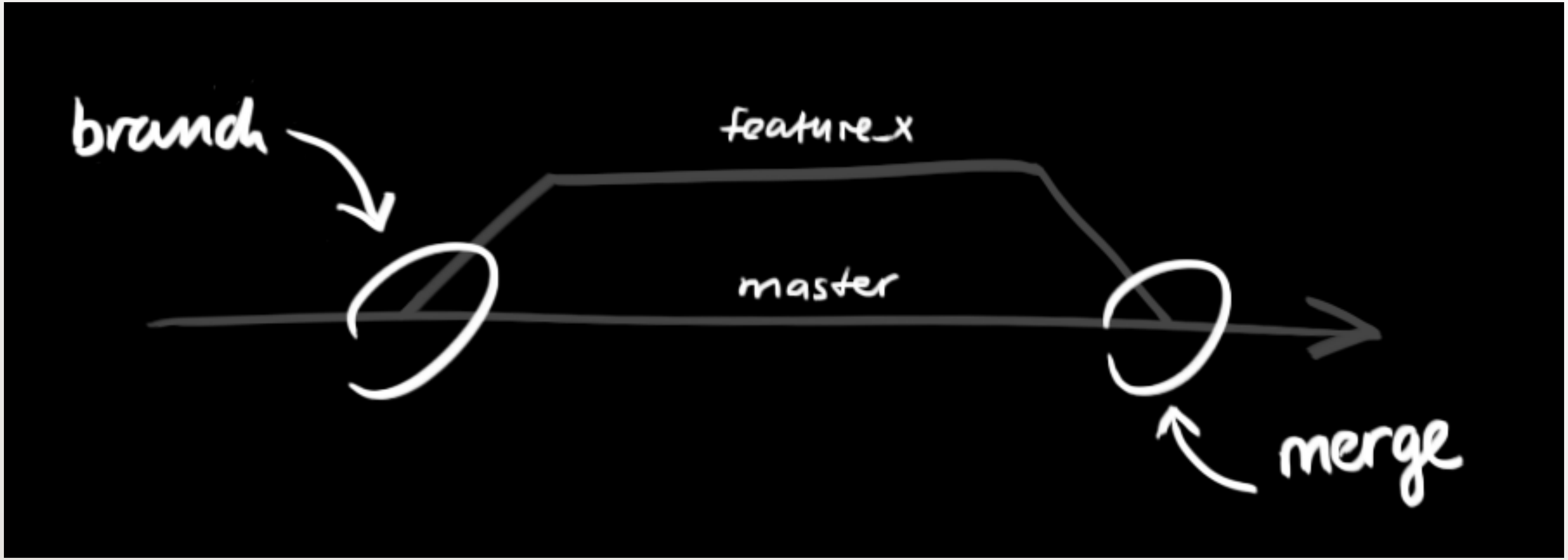
Undoing changes

- Undoing an unstaged modification:
  - *git checkout <commit> -- <filename>* or *git checkout -- <filename>*
- Undoing a staged modification
  - *git reset HEAD <filename>*
- Undoing to everything since last commit
  - git reset --hard

- Undo the change we made
  - *git checkout -- helloworld.py*
  - *git status*

Branching

- master branch, other branches



Creating and checking out a new branch

1. *git checkout -b mysuperawesomebranchthatfixesallthebugs*

Listing branches

git branch

or

git branch -v

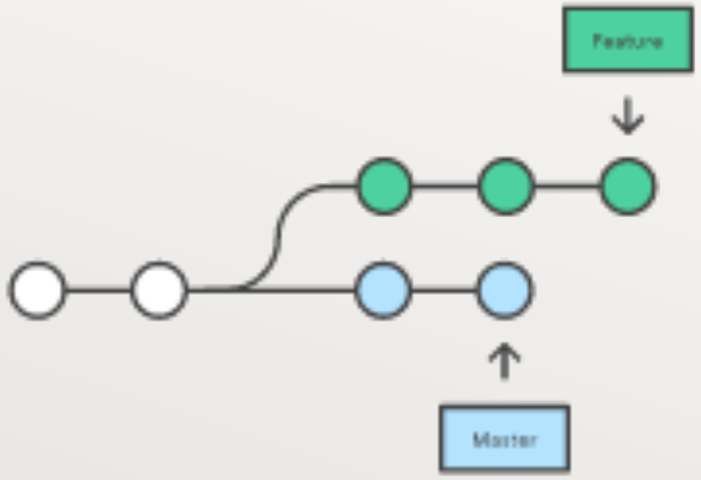
Branches are independent

- Work on a branch always

Keeping your branch updated

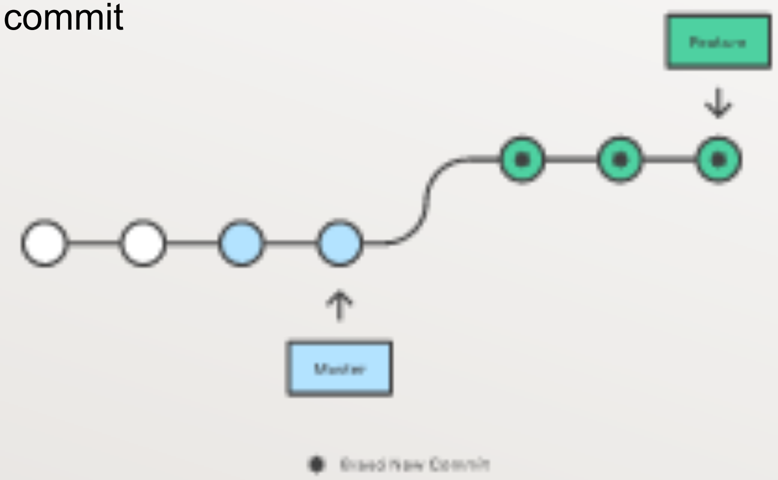
- If you took too long, or big changes in codebase happened, time to update

A forked commit history



Keeping your branch updated: rebase

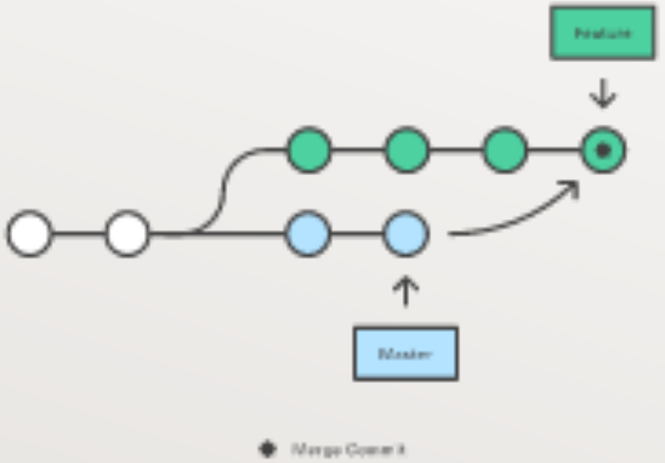
- *git rebase master*
- Moves the tail of your branch to the head of master
- Bring the base of your branch up to date
- Does not create a commit



Keeping your branch updated: merge

- Creates a commit to bring your tail up to date
- *git checkout branch*
- *git merge master*

Merging master into the feature branch



Ideology Rant: Branch Eutopia

- *Master* branch is pure goodness
- *Dev* branch is impure goodness
- Each feature/atomic change is it’s own branch
- Work on your branch, when you think (emphasis on think) you’re done
- QA your branch, and fix your mistakes (because you made some)
- Once you’re done: merge to *dev*
- Make sure that it works fine with the codebase on dev (days or weeks)
- Merge to master

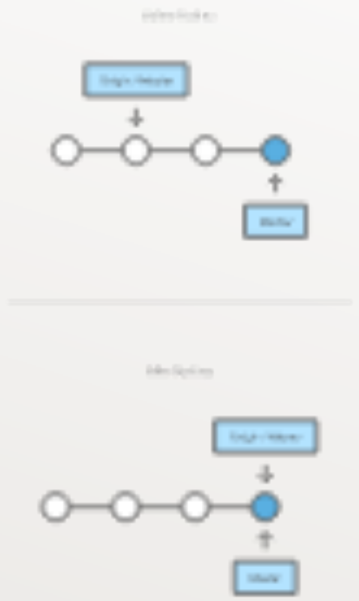
Remotes

- Remote, usually widely accessible (LAN/WAN) git repositories
- Allow multiple people to collaborate on work
- Centralized
- Bitbucket and github are two common examples
- But anyone can make git servers

Pushing

git push <where> <branch>

- *git push origin master*



Pulling

- *git pull <where>*
- Used to get the latest changes

Cloning

- *git clone <where>*
- Used to get a local copy of a remote repository

Other niceties

- Using git to deploy websites or products
- Using git to tag software releases

Other negativities

- Decentralized means no easy way to keep track of progress until push
- Requires everyone to have a full copy of the codebase
- “If things go screwy, you’re screwed”