

Currents: Coding with Cinder

Week 5: Animation with Timeline / Version Control with GitHub

Instructors

Luobin Wang (luobin@newschool.edu)

Weili Shi (weili@newschool.edu)

```
timeline().apply( ... ).finishFn( []{  
  app::console() << "This tween has finished.";  
} );
```

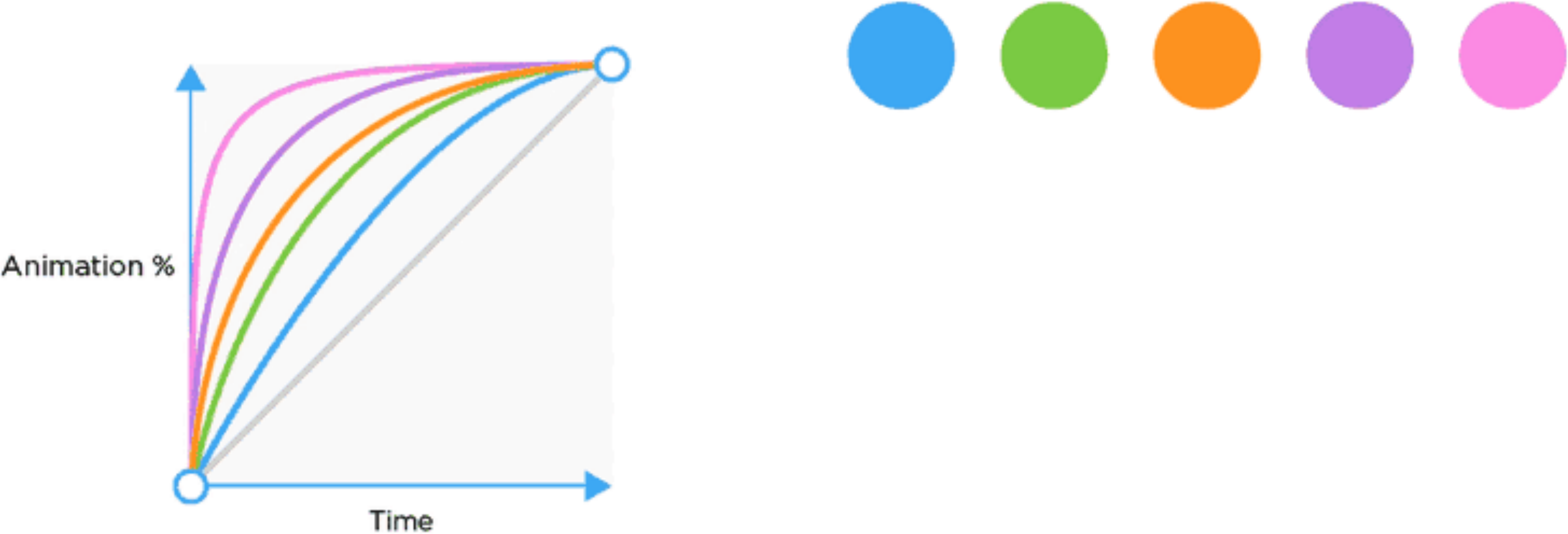
Lambda Expressions

A lambda expression can be thought of as an unnamed, inline function.

```
[capture list] (parameter list) -> return type { function body }
```

Robert Penner's Easing Functions

Easing functions specify the rate of change of a parameter over time.



Crafting Easing Curves for User Interfaces - Ryan Brownhill

timeline().apply

Apply a new interpolation

timeline().appendTo

Queue a motion

```
.easeFn(ci::yourFavoriteEasingFn)
```

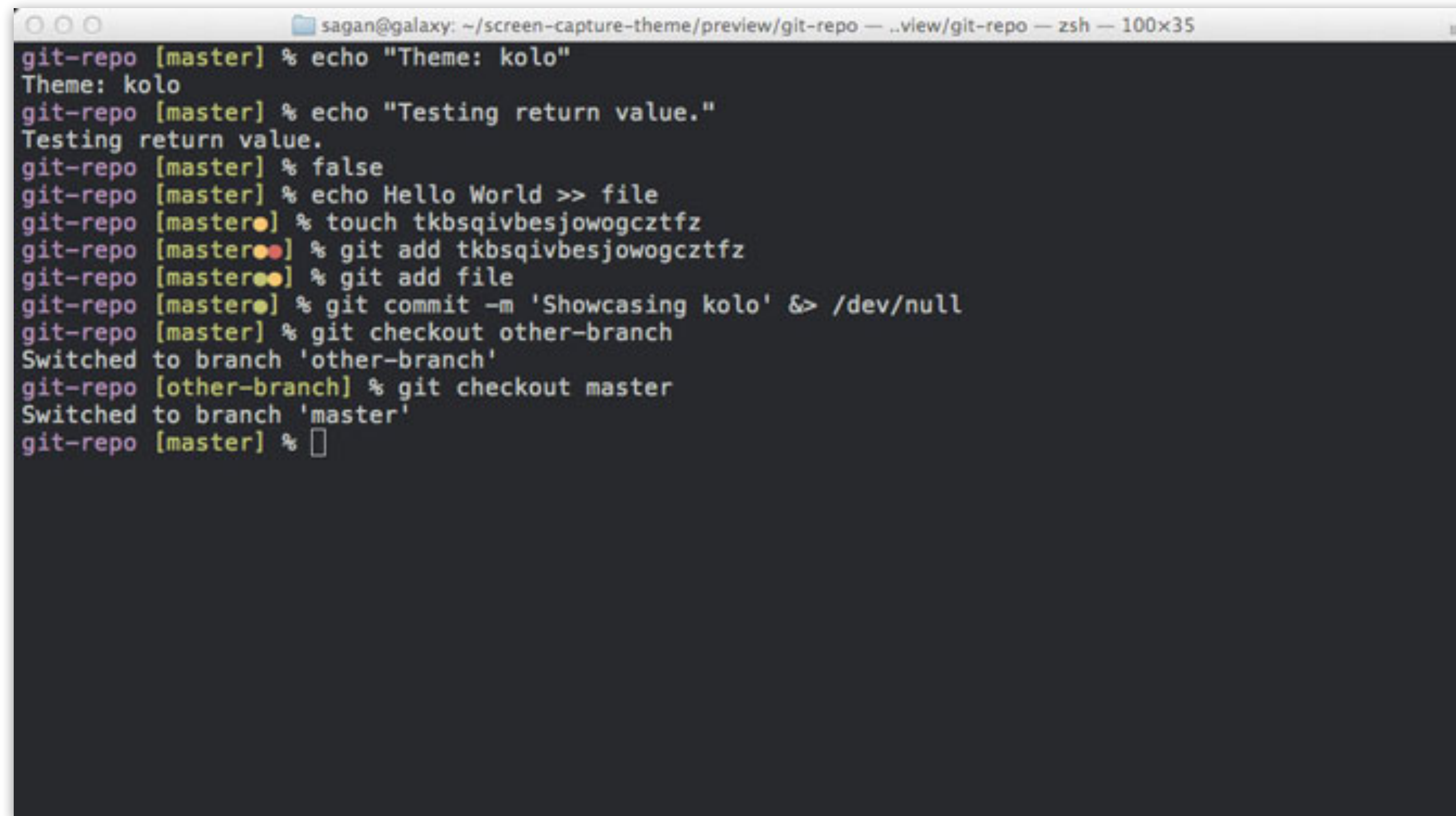
Tell the the interpolation what kind of easing to use

.delay()

Tell the timeline how long you want to delay.

Version Control with GitHub

Today will upload your homework to GitHub, and learn how to collaborate with your fellow coders using GitHub.

A terminal window titled 'sagan@galaxy: ~/screen-capture-theme/preview/git-repo — ..view/git-repo — zsh — 100x35'. The terminal shows a series of commands and their outputs. The commands are: 'echo "Theme: kolo"', 'echo "Testing return value."', 'false', 'echo Hello World >> file', 'touch tkbsqivbesjowogcztzfz', 'git add tkbsqivbesjowogcztzfz', 'git add file', 'git commit -m "Showcasing kolo" &> /dev/null', 'git checkout other-branch', 'git checkout master', and an empty prompt. The outputs are: 'Theme: kolo', 'Testing return value.', 'Switched to branch "other-branch"', and 'Switched to branch "master"'. The prompt is 'git-repo [master] %' for most commands, and 'git-repo [other-branch] %' for the checkout command.

```
sagan@galaxy: ~/screen-capture-theme/preview/git-repo — ..view/git-repo — zsh — 100x35
git-repo [master] % echo "Theme: kolo"
Theme: kolo
git-repo [master] % echo "Testing return value."
Testing return value.
git-repo [master] % false
git-repo [master] % echo Hello World >> file
git-repo [master] % touch tkbsqivbesjowogcztzfz
git-repo [master] % git add tkbsqivbesjowogcztzfz
git-repo [master] % git add file
git-repo [master] % git commit -m 'Showcasing kolo' &> /dev/null
git-repo [master] % git checkout other-branch
Switched to branch 'other-branch'
git-repo [other-branch] % git checkout master
Switched to branch 'master'
git-repo [master] %
```

Install Oh My ZSH

Go to <http://ohmyz.sh>, copy and paste the following line to your terminal:

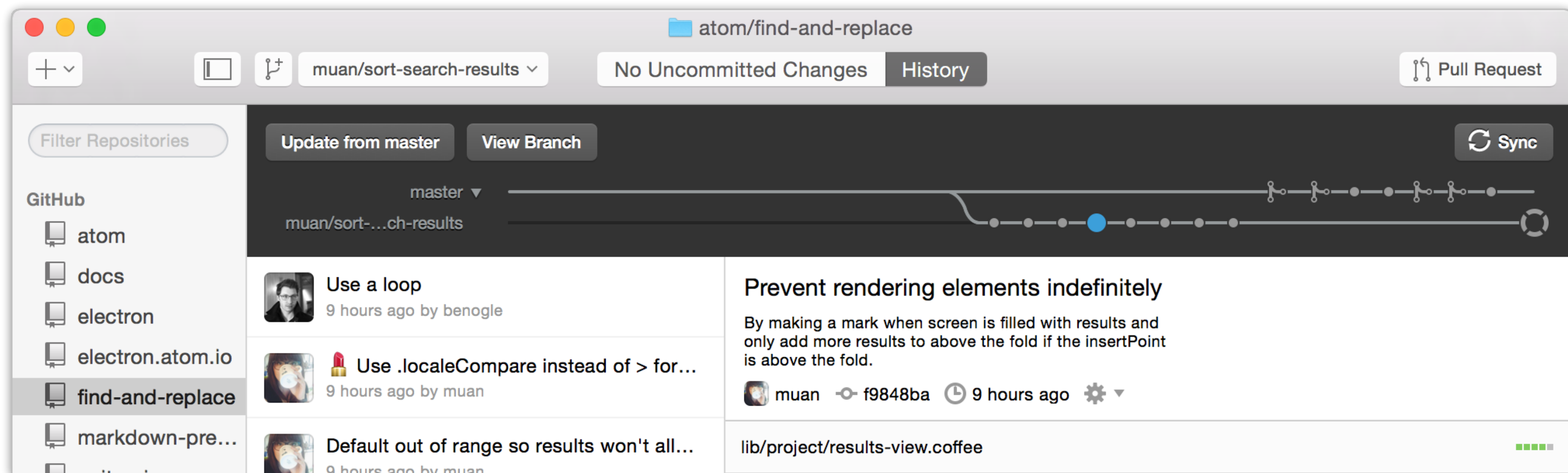
```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
```

Sign up for a GitHub account

<https://github.com/join> (If you don't have already one)

Install GitHub Desktop

<https://desktop.github.com> (Mac and Windows only)



Check if you have git and git-flow installed

```
git --version
```

```
git flow
```

If not, install them using Homebrew (Mac only)

First install Homebrew: go to <https://brew.sh>, copy and paste the following line to your terminal:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Then use Homebrew to install git and git-flow:

```
brew install git-flow
```

(Linux user: use the package manager of your Linux distribution.)

Check/set your GitHub username/email

```
git config --global user.name
```

```
git config --global user.name "Billy Everyteen"
```

```
git config --global user.email
```

```
git config --global user.email "your_email@example.com"
```

Caching your GitHub password in Git (Mac only)

Find out if the osxkeychain helper are already installed:

```
git credential-osxkeychain
```

If the osxkeychain helper isn't installed, your computer will prompt you to download it as a part of the Xcode Command Line Tools.

Tell Git to use osxkeychain helper using the global credential.helper config:

```
git config --global credential.helper osxkeychain
```

(Linux user: refer to <https://help.github.com/articles/caching-your-github-password-in-git/#platform-linux>)

Task 1: Upload your homework!

1. Go to <https://github.com/new>; create a new repository named “**firstname_currents_cinder**”.
2. Click “Set up in Desktop” to open GitHub Desktop.
(Linux user: use the alternative command-line approach shown on the same webpage.)
3. Go to <https://www.gitignore.io>; generate a .gitignore file for C++ and Xcode (or your IDE), copy the content.
4. In GitHub Desktop, go to menubar->Repository->Repository Settings...->Ignored Files, paste the .gitignore file content, and save.

(Linux user: create .gitignore manually and paste the content in.)

5. Right-click your repo name, select “Open in Terminal”, add .gitignore to stage, commit and push to GitHub.

```
git add -A
```

```
git commit -m "Add .gitignore"
```

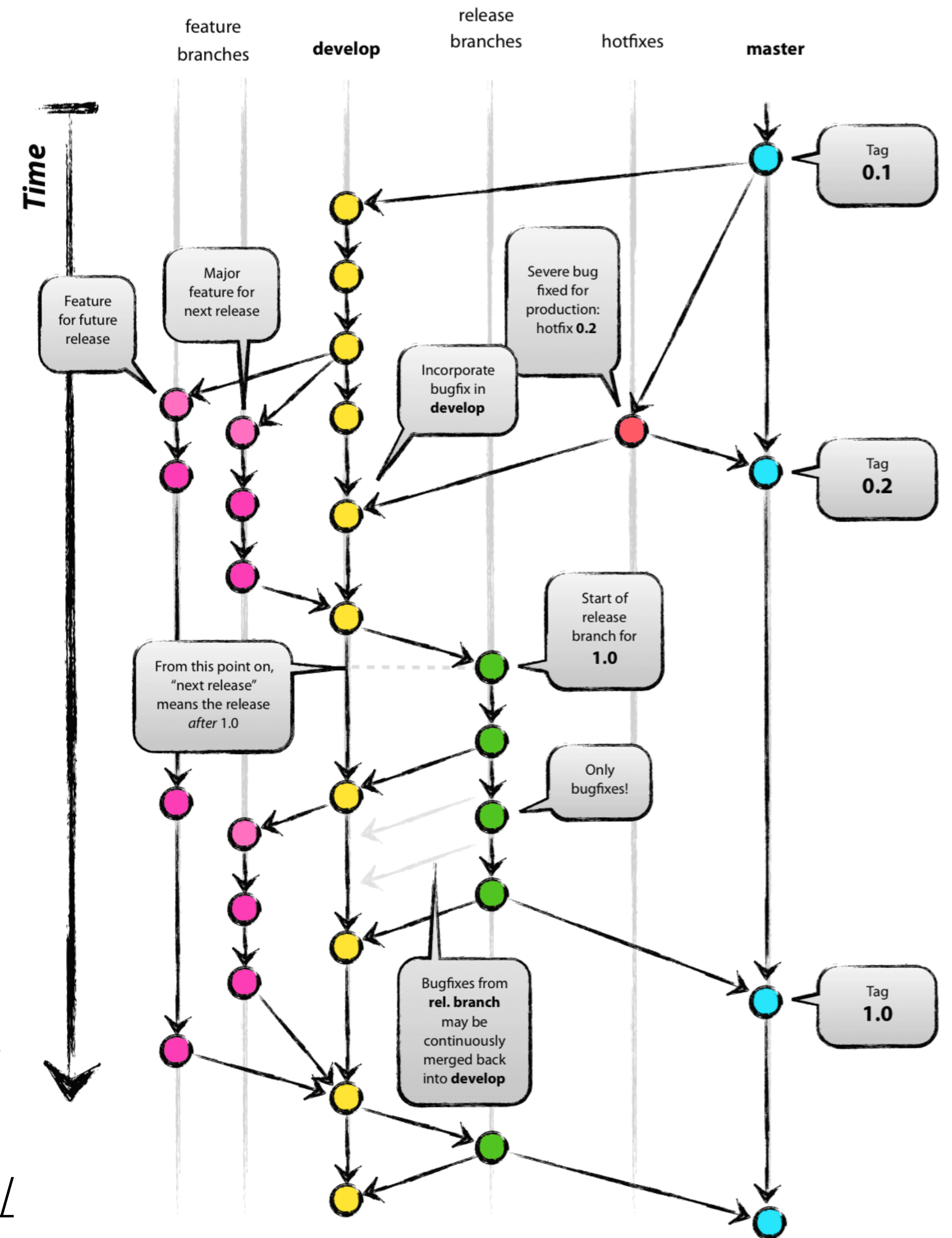
```
git push
```

6. One week per time, copy your homework of week 1~4 to the repo, commit and push.

Version Control with git-flow

Git-flow cheatsheet: <http://danielkummer.github.io/git-flow-cheatsheet/>

<http://nvie.com/posts/a-successful-git-branching-model/>



Task 2: Collaborate with your fellow programmer!

1. Pair with your peers! Form teams of two people.
2. **Member 1:** Go to <https://github.com/new>; create a new repository named “**git-flow-practice**”. Clone the repo to your computer, generate and add .gitignore file, commit and change (steps 2~5 in task 1).
3. **Member 1:** Open the repo in terminal, initialize git-flow:
`git flow init`
4. **Member 1:** Push develop branch to GitHub:
`git push --set-upstream origin develop`
Go to repo webpage, in Settings->Branches, set the default branch to develop.
5. **Member 2:** Go to the webpage of the repo; click “Clone or download”, and open the repo in GitHub Desktop.
6. **Member 2:** Open the repo in terminal, initialize git-flow:
`git flow init`
7. **Member 1:** Create a feature branch and work on it:
`git flow feature start main-cpp`
Open the repo in Finder, create main.cpp with the following content:

```
#include <iostream>
using namespace std;
int answer = 42;
int main() {
    cout << answer << endl;
    return 0;
}
```

Task 2: Collaborate with your fellow programmer!

8. **Member 1:** Commit the changes, and publish the feature to GitHub:

```
git add -A  
git commit -m "Create main.cpp"  
git flow feature publish main-cpp
```
9. **Member 1:** Open a pull request: either in GitHub desktop or on repo website, select the **feature/main-cpp** branch, and click the pull request button; create a request for merging the feature branch into **develop** branch.
10. **Member 2:** On repo website, go to “Pull requests” tab, select the pull request and review the code. When ready, click “Merge pull request” button to merge, and then delete the feature branch.
11. **Member 1:** Finish (delete) the local feature branch:

```
git checkout develop  
git pull  
git flow feature finish main-cpp
```

Task 2: Collaborate with your fellow programmer!

12. **Both:** Pull the newest code from GitHub:

```
git pull
```

13. **Member 1:** create a feature branch and work on it:

```
git flow feature start answer1
```

Modify the value of the answer variable to **43**; commit and publish:

```
git add -A
```

```
git commit -m "Modify answer to 43"
```

```
git flow feature publish answer1
```

14. **Member 2:** create a feature branch and work on it:

```
git flow feature start answer2
```

Modify the value of the answer variable to **45**; commit and publish:

```
git add -A
```

```
git commit -m "Modify answer to 45"
```

```
git flow feature publish answer2
```

Task 2: Collaborate with your fellow programmer!

15. **Member 1:** On repo website, create a pull request for the **feature/answer1** branch; merge it into **develop** branch; and delete the feature branch. In terminal, Finish (delete) the local feature branch:

```
git checkout develop  
git pull  
git flow feature finish answer1
```
16. **Member 2:** On repo website, create a pull request for merging **feature/answer2** into **develop**.
17. **Both:** Now we see a conflict. All conflicts must be resolved before merging. Both team members should sit together, click “Resolve conflicts” button, look at the conflicting parts in the code, decide which part to keep/delete, resolve all conflicts, commit changes, merge the pull request, and finally delete the feature branch.
18. **Member 2:** Finish (delete) the local feature branch:

```
git checkout develop  
git pull  
git flow feature finish answer2
```
19. **Member 1:** Update the repo according to GitHub:

```
git checkout develop  
git pull
```



Homework - modern/futuristic UI

1. Using easing and all the stuff we have learned to build motions of a interactive user interface.
2. (Bonus) Make it interactive.
3. Have all your homework (everything you made) on Github. Name your repo: **firstname_currents_cinder**

