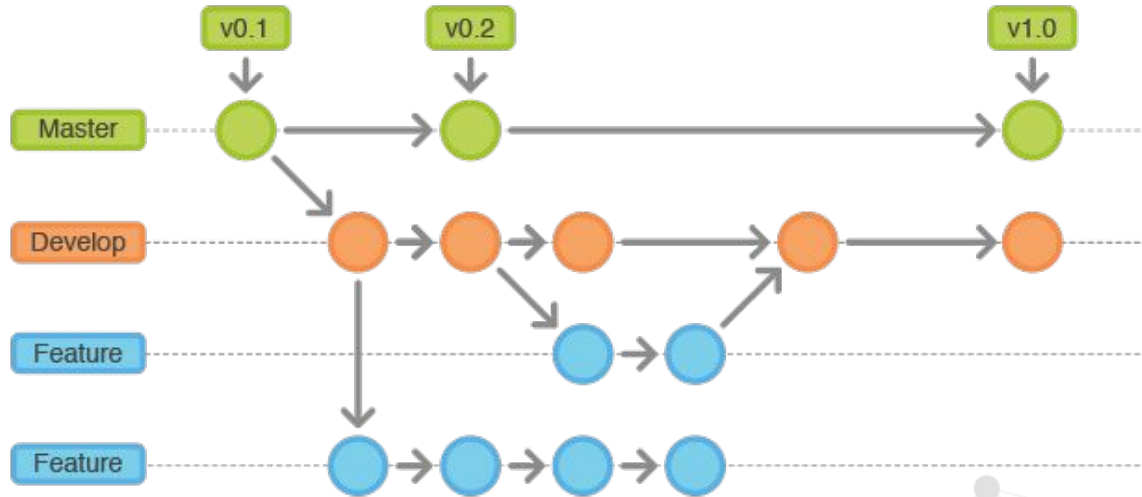




Learn Version Control the Hard Way

Alireza Samar

We're going to learn ..



MLDS

What is version control?

- Keeps track of your creative output.
- It tracks what is changed.
- It tracks who makes the changes.
- It tracks why changes were made.
- Keeps track that evolution is vital.
- Everyone needs it
 - developers / designers
 - writers / producers
 - artists / composers
- You can use it as part of team or by yourself.
- Keeping track of changes and history is the key to creative success.

Every good journey begins with a story ...



MLDS

Why version control?

- **Peace of mind:** Automatic backups
- **History:** Change-by-change log of your work
- **Friction-free undo:** For both short-term and long-term

How do teams benefits from control?

- **Synchronisation:** Easy to keep team members always up-to-date
- **Accountability:** Know who made each change and why
- **Conflict detection:** Keep the build clean every time

In fact ...
you already do version control even if you don't
use any software!

- *Thesis-Working ~~Thesis-Working~~*
- *Thesis-Shiplt ~~Thesis-Shiplt~~*
- *Thesis-Final ~~Thesis-Final~~*
- *Thesis-FinalFinalWithBugFixes*

Everything is now automated

- **Backups:** Every version is kept around
- **Change tracking:** Commit messages let you know why things changed
- **Rollback to previous versions:** It's like undo for coding
- **Labelling significant changes:** Tags/labels identify the state of the source that matches each release

A quick history

- Stand-alone and file-focused
 - **SCCS**
1972, Unix only
 - **RCS**
1982, cross-platform, text only

A quick history

- Centralized
 - **CVS**
1986, first central repository, file-focused
 - **Perforce**
1995, still the biggest repository inside Google
 - **Subversion**
2000, non-text files, track directory structure, transaction unit
 - **Microsoft Team Foundation Server**
2010, comes with MSDN subscription, tight Visual Studio integration

A quick history

- Distributed
 - **Git**
2005, created by Linus Torvalds after BitKeeper went commercial only. Broadly used in conjunction with GitHub, which offers free hosting for open-source projects.
 - **Mercurial**
2005, also created in response to BitKeeper change

Essential version control concepts

- **Repository** aka database
 - where your files and their history is stored
- **Working set**
 - the current state of the files as stored on your local machine
- **Add**
 - insert new files from working set into the repository

Essential version control concepts

- **Check-in / Commit**

- copy changes from working set to repository

- **Check-out / Update**

- copy changes from repository to working set

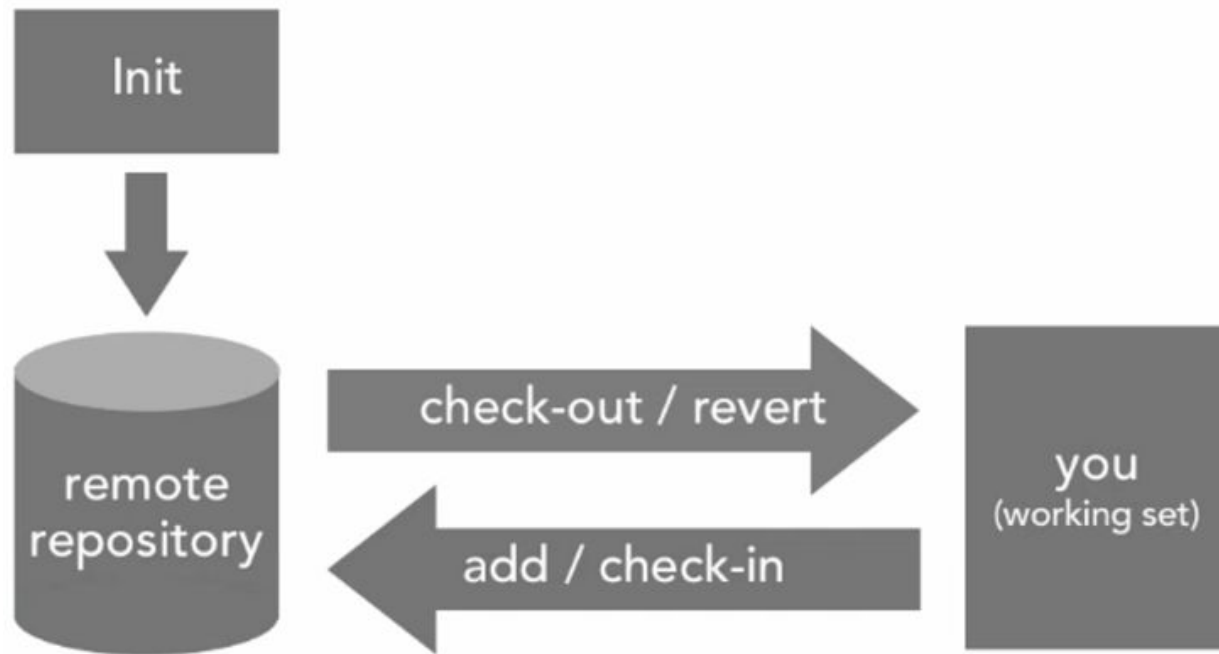
- **Tag / Label**

- mark the current state of the repository for future checkout

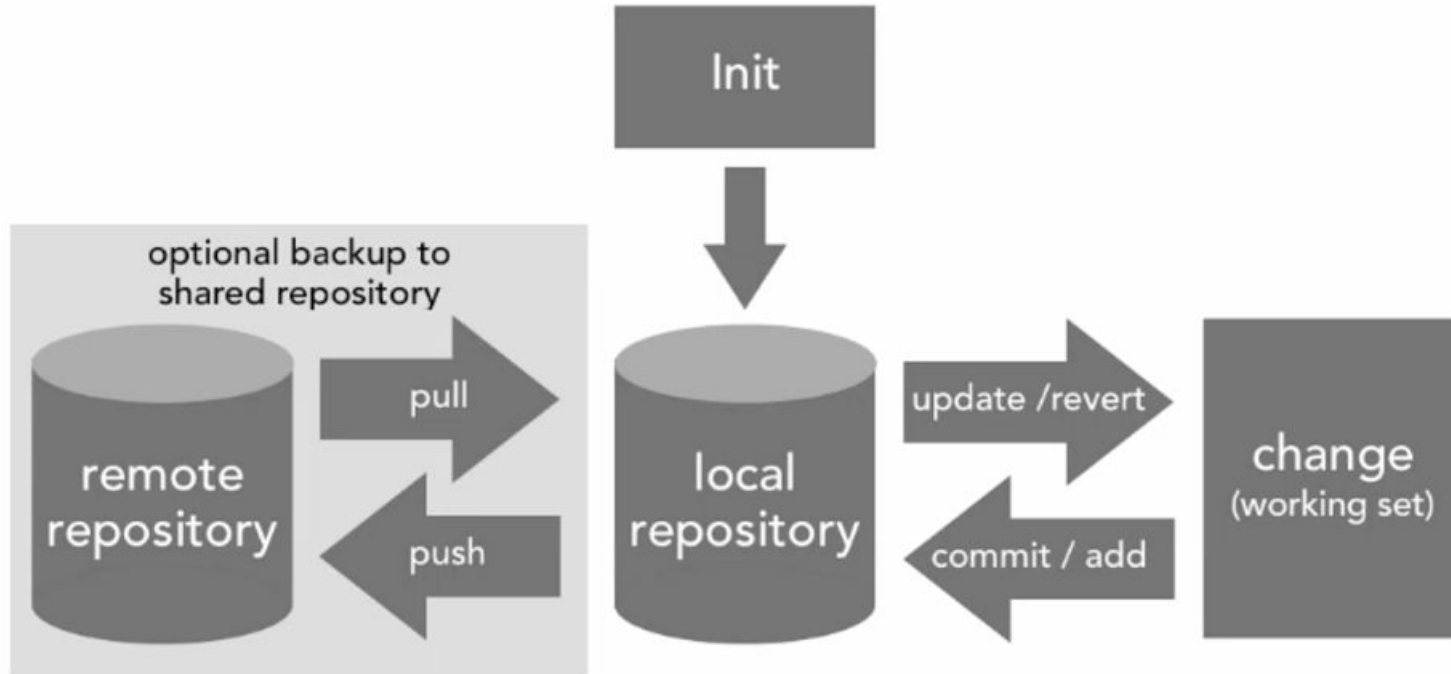
Distributed repository systems only

- **push / export**
 - send changes from one repo to another
- **pull / import**
 - update your working set with updates
- **Tag / Label**
 - name a specific state of a repo
- **branch / fork**
 - make a clone of a repo
- **Merge**
 - integrate your branch (clone) back into the original repo

Centralized vs. Distributed



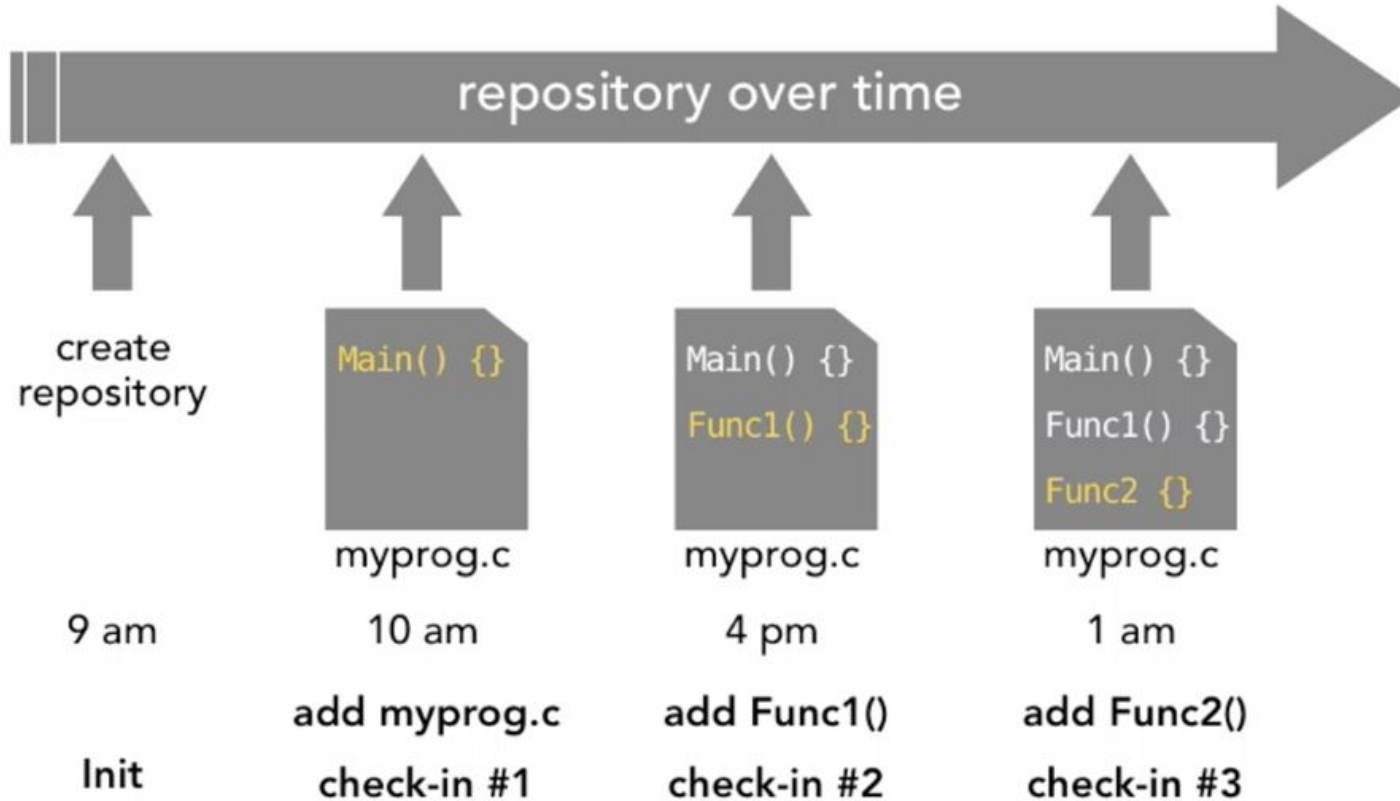
Centralized vs. Distributed



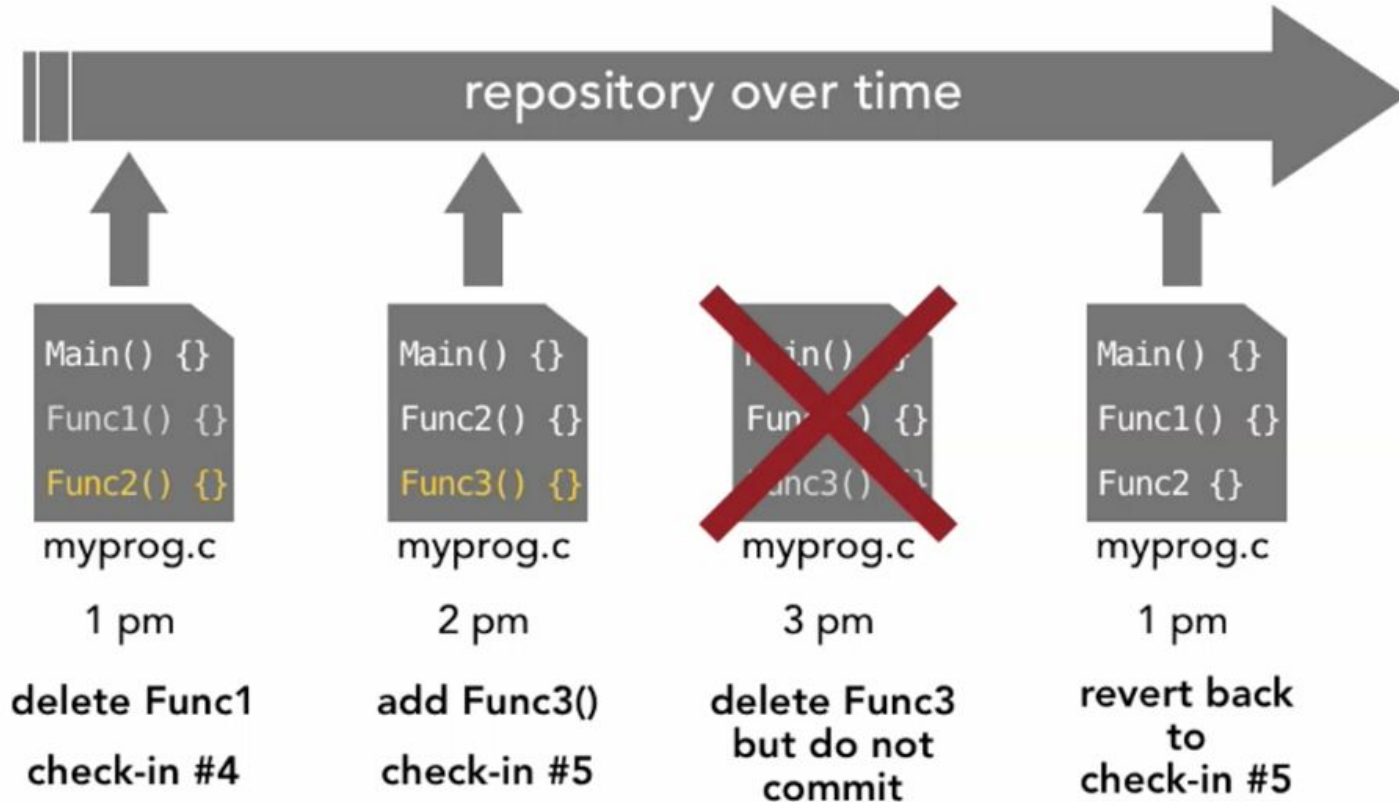
Git

- Type: Distributed
- Free with commercial hosting options
- Open-source free bits: <http://git-scm.com>
- Free and commercial hosting for open-source and closed source:
<http://github.com>

Git in action



Git in action



Let's do it together ...

1. `git --version` (make sure Git installed properly)
2. `mkdir fsvc` (create your directory)
3. `cd fsvc` (change the path to your directory)
4. `git init` (setup Git)
5. `dir /a` (check the Git hidden database file)
6. `git status` (make sure it initialised properly)

* if you don't have git installed, get it from [here](#) for Windows

Let's do it together ...

7. `notepad myprog.py` (create a file, add `"Main () {}"` and save it)
8. `git add *` (to tell Git to track new files)
9. `git commit -a -m "Add myprog.py"`
10. Error?
 - a. `git config --global user.email "you@example.com"`
 - b. `git config --global user.name "Your Name"`
11. `git commit -a -m "Add myprog.py"`
12. `git status` (to verify your success commit)

Let's do it together ...

13. `git log -p` (ask Git to tell you the history)
14. `notepad myprog.py` (let's reopen the file, add `"Func1 () {}"` and save it)
15. `git commit -a -m "Add Func1()"`
16. `git log -p` (check the changes you've made)
17. `notepad myprog.py` (one more time, let's reopen the file, add `"Func2 () {}"` and save it)
18. `git commit -a -m "Add Func2()"`
19. `git log -p` (check the changes you've made)

You're almost there ...

- 20. `git log --oneline --all` (ask Git for shorter version of your history)
- 21. `git diff 6a3e577 7555e83` (ask Git to show you the differences, take note that identifiers are different for you Git!)

Let's make some Oops!

22. `notepad myprog.py` (open the file, delete “Func1 () {}” and save it)
23. `git diff` (check what's in repo and what's in working set)
24. `git commit -a -m “Delete Func1()”`
25. `notepad myprog.py` (let's reopen the file, add “Func3 () {}” and save it)
26. `git commit -a -m “Add Func3()”`
27. `git log --oneline --all`

Let's make some Oops!

28. `notepad myprog.py` (open the file, delete “Func3 () {}” and save it) (we did it in our working set, didn't commit yet)
29. `type myprog.py` (but we didn't meant to do that!)
30. `git diff HEAD` (check the difference between repo and working set)
31. `git checkout myprog.py` (revert back the mistake we've made)
32. `type myprog.py` (and it's back again!)
33. `git diff HEAD / git diff / git status` (you can see there is no difference)

You got it all!

<https://desktop.github.com>

The screenshot shows the GitHub Desktop application window. The title bar indicates the current project is 'atom/find-and-replace'. The interface includes a sidebar on the left with a 'Filter Repositories' search bar and a list of repositories: atom, docs, electron, electron.atom.io, find-and-replace (selected), markdown-preview, and unity-ui. The main area is divided into three sections. The top section shows the current branch 'master' and a visual commit history with a blue dot indicating the current commit. The middle section displays a list of recent commits, with the selected commit being 'Default out of range so results w...' by muan, 2 hours ago. The bottom section shows a code diff for the file 'lib/project/results-view.coffee', highlighting changes in the 'class ResultsView extends ScrollView' section.

atom/find-and-replace

+ ▾

muan/sort-search-results ▾

No Uncommitted Changes

History

#320

Filter Repositories

Update from master

View Branch

Sync

master ▾

muan/sort-...ch-results

GitHub

- atom
- docs
- electron
- electron.atom.io
- find-and-replace
- markdown-preview
- unity-ui

Use a loop
1 hour ago by benogle

Use .localeCompare instead...
2 hours ago by muan

Default out of range so results w...
2 hours ago by muan

Remove check if should render...
2 hours ago by muan

Default out of range so results won't all get prepended
muan bbb4a18 2 hours ago

lib/project/results-view.coffee

```
...  ...  @@ -71,7 +71,7 @@ class ResultsView extends ScrollView
71  71      for filePath in paths[@lastRenderedResultIndex..]
72  72          result = @model.getResult(filePath)
73  73          listItems = @children()
74  74          insertPoint = 0
```

Thanks!

Machine Learning for Data Science Interest Group
Advanced Informatics School
Universiti Teknologi Malaysia

@utmmls
ais.utm.my/mls

February 2017



MLDS