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.

http://www.xkcd.com
Why git?  
Local Git  
Remote Git  
Git branches  
Best practices  
Culture  

Software Development  
Entrepreneurship
Why git?

Software Development Entrepreneurship
Why git?

Local Git

Remote Git

Git branches

Best practices

Culture

Software Development Entrepreneurship
Why git? Local Git Remote Git Git branches Best practices Culture

Software Development Entrepreneurship
Why git? Local Git Remote Git Git branches Best practices Culture

Software Development Entrepreneurship
Why git? Local Git Remote Git Git branches Best practices Culture

Software Development Entrepreneurship
Git helps you to:

- continuously keep backups of your work;
- restore the entire state of your project to a previous version;
- undo specific changes to some files – or even part of files – without affecting others;
- easily share your progress with other people;
- synchronise your work across multiple devices;
- experiment with a new feature without affecting existing work;
- maintain multiple versions of a product;
- and more!
<table>
<thead>
<tr>
<th>Why git?</th>
<th>Local Git</th>
<th>Remote Git</th>
<th>Git branches</th>
<th>Best practices</th>
<th>Culture</th>
</tr>
</thead>
</table>

**Actions:**

Software Development Entrepreneurship
Actions: figure out which files should be on version control
Why git?  Local Git  Remote Git  Git branches  Best practices  Culture

Actions: `git add *.txt *.java images/ ; git commit`
Actions: replace all occurrences of “magenta” by “pink”
Actions: `git commit -a`
Actions: add a new source file and change some others
Actions: git add newfile.java ; git commit
Actions: start working on a new feature
Actions: realise it was a bad idea
Actions: `git reset --hard`
Actions:
<table>
<thead>
<tr>
<th>Why git?</th>
<th>Local Git</th>
<th>Remote Git</th>
<th>Git branches</th>
<th>Best practices</th>
<th>Culture</th>
</tr>
</thead>
</table>

Actions: *update documentation*
Actions: git commit -a
Actions: decide you want to see an earlier version
Actions: git checkout de337dc
Actions: `git checkout master`
Actions: realise that the colour change was a bad idea
Actions: git revert de337dc
Actions: realise that you were drunk during the last two commits
Actions: `git reset 252137e --hard` (be very careful!)
Actions:
You can also:

- view a graphic description of your commits (like given here)
- “stage” changes gradually
- view files in earlier versions (git show a62c16e:file1.txt)
- recover only a single file (git checkout a62c163 -- file1.txt)
- view differences between a current and prior version of a file
- stage changes gradually
Actions:
Actions: git push
Actions: git pull
Actions: Martha updates the database handlers.
Actions: `git commit -a`
Actions: `git push`
Actions: git pull
Actions: Martha decides that the colour change was a bad idea.
Actions: git revert de337dc
Actions: Harry changes the visualisation module.
Actions: `git push`
**Actions:** `git commit -a`
Actions: `git pull`
Actions: Harry handles the merge conflicts (if necessary).
Why git?  
| Local Git | Remote Git | Git branches | Best practices | Culture |

Actions: `git push`

Software Development  
Entrepreneurship
Actions:
Actions: start work on the network
Actions: `git commit -a`
Actions: decide to work on a high-priority database change
Actions: git checkout -b foo de337dc
Actions: make some changes to the database
Actions: git commit -a
Actions: complete changes to the database
Actions: `git commit -a`
Why git? Local Git Remote Git Git branches Best practices Culture

Actions: git checkout master
Actions: improve network negotiation
Actions: `git commit -a`
Actions: fix last of the network bugs
Actions: git commit -a
Actions: `git merge foo`
master

foo

Actions:
Actions: Harry decides to start working on a new feature.
Actions: `git branch DB`
Actions: Harry changes the table structure
Actions: `git commit -a`
Actions: Collaborators make their own changes to the project.
Actions: Harry moves database interactions to a separate class
Actions: git add DBManager.java ; git commit -a
Actions: Collaborators make more changes and ask Harry to look.
Actions: git checkout master
Actions: git pull
Actions: Harry makes a minor update.
Actions: `git commit -a`
Why git? Local Git Remote Git Git branches Best practices Culture

Actions: git push
Actions: git checkout DB
Actions: **Harry completes changes to the database.**
Actions: git commit -a
Actions: git checkout master
Actions: `git merge DB`
Actions: git push
You can also...

- push branches onto the server (`git push -u origin <branchname>`)  
- view which branches you have, which is active (HEAD), etc.  
- keep a branch up-to-date with another without merging (`git rebase master`)  
- delete whole branches (`git branch -d <branchname>`)  
- change branch names; make another branch master (but be careful!)
<table>
<thead>
<tr>
<th>Why git?</th>
<th>Local Git</th>
<th>Remote Git</th>
<th>Git branches</th>
<th>Best practices</th>
<th>Culture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bad workflow**

```
master
```

- Initial commit
- First release
- New function with bug
- Bug fixed
- Minor improvement
- Second release
Bad workflow

Initial commit

master
Bad workflow

Initial commit
First release
Bad workflow

<table>
<thead>
<tr>
<th>Initial commit</th>
<th>a1</th>
</tr>
</thead>
<tbody>
<tr>
<td>First release</td>
<td>c9</td>
</tr>
<tr>
<td>New function with bug</td>
<td>a6</td>
</tr>
</tbody>
</table>
Bad workflow

Initial commit

First release

New function with bug

Bug fixed

Workflow

master

a1

c9

a6

d2
Bad workflow

- Initial commit
- First release
- New function with bug
- Bug fixed
- Minor improvement
Bad workflow

- Initial commit
- First release
- New function with bug
- Bug fixed
- Minor improvement
- Second release
Good workflow

- master
- develop
- feature
Good workflow

Initial commit
Good workflow

Initial commit

Working version
Good workflow

- Initial commit
- Working version
- First release

Diagram:
- Master branch
- Develop branch
- Feature branch
- Commits:
  - a1
  - c9
  - 65

Software Development Entrepreneurship
Good workflow

Initial commit
Working version
First release
New function with bug
Good workflow

- Initial commit
- Working version
- First release
- New function with bug
- Bug fixed
Good workflow

Initial commit

Working version

First release

New function with bug

Bug fixed

Update working version
Good workflow

- Initial commit
- Working version
- First release
- New function with bug
- Bug fixed
- Update working version
- Minor improvement
Good workflow

Initial commit
Working version
First release
New function with bug
Bug fixed
Update working version
Minor improvement
Second release
Try it out!!

• make a local git repository (possibly: one of those you played with before)
• create an empty repository on github
• push your repository to the server
• make branches and push those to the server, too
• delete branches on the server
• (fork and) clone someone else’s repository
• generate and resolve merge conflicts
• any remaining questions... try it yourself :)}
General advice

- do not mess with remote history!
- use a development branch next to a release branch
- use a separate branch feature/pick-the-name for each feature
- append .git to directory of remote repository (e.g., project.git)
- all files in .gitignore will be ignored (e.g., *.sw? for swap files and *~ for editor backup files)
- keep an eye on the repository structure through git gui (or an alias)
Some final notes on culture
Some final notes on culture

Direct negative feedback

Indirect negative feedback
Some final notes on culture
Some final notes on culture

- People from different cultures have different ways of communicating.
- Be aware of your own cultural habits and assumptions.
- If you aren’t sure how something is meant, ask.
- Have a discussion about your team approach to:
  - feedback
  - management
  - sharing ideas
  - resolving conflicts
  - building the team
  - timing and deadlines