

Review Questions

This document contains review questions for Git basics.

Instructions

Overview

* **Problem**: Consider the example from the lecture, where we created a branch for our data analysis. Why did we create a new branch for this? Why *not* simply do this on `master`?

* **Solution**: Doing the work directly on `master` would make it harder to keep that work organized. Branches let us keep our work sandboxed and organized.

* **Problem**: Write two advantages to creating branches instead of working directly on `master`.

* **Solution**:

1. We can isolate our experiments to a single branch—if we break something on our `data_analysis` branch, we at least know that the code on `master` still works.

2. We can focus on writing *one* new feature at a time, instead of having work for a handful of different features and bugfixes in a single branch.

Branching

For the problems below, consider the following commit history:

```
```bash
(master) | [m1] -> [m2] -> [m3] -> [m4]
```
```

* **Problem**: Draw a new branch, called `plotting_data`. It should branch from the second commit to `master`.

* **Solution**:

```
```bash
(master) | [m1]-[m2]
 | \
(plotting_data) | \-[pd1]-[pd2]
```
```

* **Problem**: When you first create `plotting_data`, are the files on that branch the same as the files in `[m1]`? In `[m2]`? Why, or why not?

* **Solution**:

The files on `plotting_data` are the same as the files in `[m2]`.

This is because we created and checked out `plotting_data` while we were on `[m2]`. This means our files will look like they did when we last committed to `[m2]`, but that Git tracks any further changes to files on `plotting_data`, *not* `master`.

* **Problem**: Add two commits to the `master` branch.

* **Solution**:

```
```bash
(master) | [m1]-[m2]-[m3]-[m4]
 | \
(plotting_data) | \-[pd1]-[pd2]
```
```

* **Problem** Add two commits to the `plotting_data` branch, named `[pd1]` and `[pd2]`.

* **Solution**

```
```bash
(master) | [m1]-[m2]-[m3]-[m4]
 |
(plotting_data) | \-[pd1]-[pd2]
```
```

* **Problem**: Are the files in `[pd1]` and `[m3]` the same? Why, or why not?

* **Solution**: No. The code in `[pd1]` and `[m3]` are *not* the same. Any two given branches should contain *different* work, so `[m3]` probably contains a patch or bugfix totally unrelated to `[pd1]`.

Merging

* **Problem**: Merge `[pd2]` with `master`.

* **Solution**:

```
```bash
(master) | [m1]-[m2]-[m3]-[m4] - --[m5]
 |
(plotting_data) | \-[pd1]-[pd2]-/
```
```

* **Problem**: Explain how this merge changes the files in `master`.

* **Solution**: `master` now has the most recent changes to `clean_data.py` made on *either* the `master` or `plotting_data` branch. It will also have the new files added in the `plotting_data` branch.

* **For the problems below, consider the following history.**

```
```bash
(master) | [m1]-[m2]-[m3]-[m4] - - - - -[m5]
 |
(plotting_data) | \-[pd1]-[pd2]-/ (M)
```
```

* Assume `[m4]` on `master` updates `clean_data.py`, but doesn't change the directory structure.

* Assume the root project directory looks as follows at each commit:

```
```bash
[m4]
root/
|_ analyze_data.py
|_ clean_data.py
|_ output/
|_ cleanedRideData.csv
|_ Resources/
|_ rideData.csv

[pd2]
root/
|_ analyze_data.py
|_ clean_data.py
|_ helpers.py
|_ plot_data.py
|_ output/
|_ cleanedRideData.csv
|_ plots.pdf
```
```

```
[pd2]
root/
  |_analyze_data.py
  |_clean_data.py
  |_helpers.py
  |_plot_data.py
  |_output/
    |_cleanedRideData.csv
    |_plots.pdf
  |_Resources/
    |_rideData.csv
  ...
```

* **Problem**:

1. When we merge ``master`` and ``plotting_data``, which version of each file do we get?
2. Draw the directory structure for the last commit to ``master``—after the merge—and label each file with the branch it comes from. Assume that the only files changed on ``plotting_data`` were ``helpers.py`` and ``plot_data.py``.

* **Solution**:

```
```bash
[m5]
root/
 |_analyze_data.py (master)
 |_clean_data.py (master)
 |_helpers.py (plot_data)
 |_plot_data.py (plot_data)
 |_output/ (mixed)
 |_cleanedRideData.csv (master)
 |_plots.pdf (plot_data)
 |_Resources/ (master)
 |_rideData.csv
 ...
```

- - -

# # Branch Demo

## ## 0. Getting the Repo

Before we can work with Git, we must either **create a new repository**, or **clone one from GitHub**.

Note that, in the examples below, we use `git status` before every `git commit`. This is a best practice that helps ensure a deliberate commit history. For brevity's sake, this line will be omitted in future files, **but assume we've always run `git status` before any `git commit`**.

### ### Clone from GitHub

If someone has already shared a repository on GitHub, you can **clone** it to your local machine with `git`.

```
```bash
# Clone an existing repo.
git clone <repo_url>
# Navigate into newly created repo directory
cd <repo_name>
```
```

## ## 1. Add Files

Next, we simply develop as normal, and `commit` our changes whenever we make significant progress.

In general, it's best to **commit early** and **commit often**. Frequent snapshots ensure you'll never be far away from a "last working version".

```
```bash
# Create a file, called clean_data.py
touch clean_data.py

# Add and commit clean_data.py...
git add clean_data.py
git status
git commit -m "First commit."

# Add cleanup code to clean_data.py...
git add clean_data.py
git status
git commit -m "Clean up provided data."

# Add code to export clean data...Note that `add .` adds
# everything in the current folder
git add .
git status
git commit -m "Export clean data as CSV."
```
```

```
Add code to export clean data...Note that `add .` adds
everything in the current folder
git add .
git status
git commit -m "Export clean data as CSV."
```
```

3. Merge

Once we've developed and tested the changes on our `data_analysis` branch, we can include them in `master` by **merging** the two branches.

```
```bash
Move back to master
git checkout master

Merge changes on data_analysis with code on master
git merge data_analysis

Delete the data_analysis branch
git branch -d data_analysis
```
```

N.b., deleting the `data_analysis` branch isn't necessary, but it's best practice to prune unneeded branches.

```
# Create a new directory, and initialize a Git repo inside of it.
mkdir git_practice
cd git_practice

# Create an `hello.py`. In the page body, put a heading with the text `Welcome`, and a paragraph with Lorem text.
touch hello.py

# Add and commit your `hello.py`.
git add hello.py
git commit -m "First commit."

# Create and checkout a new branch, called `helpers`.
git branch helpers
git checkout helpers
# Or: git checkout --branch helpers

# Add greet function to helpers.py
git add helpers.py
git commit -m "Add greet function to helpers.py."

# Update hello.py
git add hello.py
git commit -m "Refactor hello.py to use greet function."

# Move back to your `master` branch.
git checkout master

# Merge `master` with your `helpers` branch.
git merge helpers

# Delete your `helpers` branch.
git branch -d helpers
```

File Edit View Language

```

1 # Switch to master
2 git checkout master
3
4 # Set origin/master as the default branch to push to
5 # when on master
6 git push origin master -u
7
8 # Now, when we're on master, this is the same as:
9 # 'git push origin master'
10 git push
11
12 # Switch to push_example
13 git checkout push_example
14
15 # Set origin/push_example as the default branch to push to
16 # when on push_example
17 git push -u origin push_example
18
19 # Now, when we're on push_example, this is the same as:
20 # git push origin push_example
21 git push
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

File Edit View Language

```

1 # Create a new GitHub repo to associate with the local Git repo you've been working on thusfar. Add it as the `origin` remote.
2 git remote add origin <github_repo_url>
3
4 # Next, checkout your `master` branch, and push it to GitHub.
5 git checkout master
6 git push origin master
7 # Or: git push -u origin master
8
9 # Create and checkout a new branch
10 git branch add_gitignore
11 git checkout add_gitignore
12 # Or: git checkout -b add_gitignore
13
14 # Change something in your project-this adds a .gitignore file
15 echo ".DS_STORE" > .gitignore
16 git add .gitignore
17 git commit -m "Add .gitignore file."
18
19 # Push this branch to GitHub
20 git push -u origin add_gitignore
21
22 # Checkout `master`, and merge it with the branch you just created.
23 git checkout master
24 git merge add_gitignore
25
26 # Push the updated `master` branch to GitHub.
27 git push
28 # Or: git push origin master
29

```

1 # Branching in Git

2
3 This document contains a branching workflow cheatsheet.

5 ## Branching Workflow

- 6
- 7 1. Create a branch: ``git branch <branch_name>``
 - 8 2. Checkout the new branch: ``git checkout <branch_name>``
 - 9 3. Use ``git add`` and ``commit`` as normal.
 - 10 4. When you want to push your branch to GitHub, do: ``git push origin <branch_name>``
 - 11 5. When you want to merge your new branch into master, do the following four steps:
 - 12 1. ``git checkout master``
 - 13 2. ``git pull origin master``
 - 14 3. ``git merge <branch_name>``

15
16 Note that you can do ``git checkout --branch <branch_name>`` to create a new branch *and* immediately check it out with a single command.

17
18 - - -

1 # Branching Recipes

3 ## Initialize Repository

4 To initialize a Git repo from the command line:

```
5  
6 ```  
7 git init  
8 ```
```

11 ## Display All Branches

12 To display all branches:

```
13  
14 ```  
15 git branch  
16 ```
```

19 ## Display All Branches, with Metainformation

20 To display all branches, including information about the latest commit on each branch.

```
21  
22 ```  
23 git branch --verbose
```

```
24  
25 # Or, as shorthand  
26 git branch -v  
27 ```
```

30 ## Create a new branch

31 To create a new branch, but *not* switch to it:

```
32  
33 ```  
34 git branch <branch_name>  
35 ```
```

38 ## Move to an Existing Branch

39 To move to a branch that already exists:

```
40  
41 ```  
42 git checkout <branch_name>  
43 ```
```

44


```

45
46 ## Create & Checkout New Branch Simultaneously
47
48 To create and immediately checkout a new branch:
49
50 ```
51 git checkout --branch <branch_name>
52
53 # Or, as shorthand
54 git checkout -b <branch_name>
55 ```
56
57 ## Add a Remote for Push/Pull
58
59 To add a remote to push to/pull from:
60
61 ```
62 git remote add <remote_name> <remote_url>
63 ```
64
65 For example, to add a GitHub repo called `remote_example` as `origin`, do:
66
67 ```
68 git remote add origin github.com/peleke/remote_Example
69 ```
70
71 ## Merge Branches
72
73 To merge a branch into the branch you're currently on:
74
75 ```
76 git merge <branch_name>
77 ```
78
79 For example, doing:
80
81 ```
82 # While on `master`...
83 git merge new_feature
84 ```
85
86 ...Will merge the `new_feature` branch into `master`.
87
88
89 ## Push to GitHub
90
91 To push changes for a branch to GitHub:
92
93 ```
94 git push origin <branch_name>
95 ```
96
97 To set a default upstream branch, add the `-u` flag. For example, doing this:
98
99 ```
100 # While on the `master` branch...
101 git push -u origin master
102 ```
103
104 ...Makes it such that we can just write `git push` instead of `git push origin master`.
105
106 ## Aliases
107
108 **Aliases** allow you to give your own names to various Git commands, which allows us to save typing.
109
110 ```bash
111 # Run the following to install common aliases
112
113 git config --global alias.co checkout
114 git config --global alias.br branch
115 git config --global alias.ci commit
116 git config --global alias.st status
117 ```
118
119 This allows us to write `git ci -m "First"` instead of `git commit -m "First"`.
120
121 - - -

```