Version Control with Git

Jamie Fairbrother$^1$

$^1$Centre for Transport and Logistics, Lancaster University

Joint STORC and Transport and Logistics Exchange, 2018-02-15
Version Control and Git
Why use Version Control?

- When writing code or documents, it is often desirable to revert back to a previous state.
- Keeping multiple versions of files under different names can become unmanageable.
- When working in collaboration it is important to be able to know what changes are made, why and who made them.
What is Version Control?

- Version control systems keep track of changes made to a set of files:
  - It allows users to recover previous versions of files
  - Facilitates collaboration by allowing users to merge each others changes into their files
- Version control works by storing a sequence of “snapshots” of the directory under version control
- Supports parallel lines of development to allow users to work simultaneously on different features
Git

- Git is a popular open source cross-platform version control system
- Created in 2005 by Linus Torvalds for the development of the Linux kernel
Working tree and repository

• A project under git version control consists of the following elements:

Working tree  Directory containing all files in project

Repository  Database containing history of project
Model of Git

- Server
- Remote repository
  - Push changes
  - Fetch changes
- Local repository
  - Commit changes
- Working tree

Contributor 1
Contributor 2
Remote Hosts

- GitHub
- GitLab
- Bitbucket
Version Control and Git

Basic Usage

Demonstrations and Tips

Interfaces

Terminal

Inside IDE/Text Editor (e.g. Emacs, RStudio)

Dedicated GUI (e.g. SmartGit)
Basic Usage
Installation and Initial Set-up

- On Ubuntu-based systems, git can be installed with the following command:
  sudo apt-get install git
- In Windows, git can downloaded from the following page:
  https://git-scm.com/download/win
- Before using git, you must configure your name and email address:
  git config --global user.name "Jamie Fairbrother"
  git config --global user.email fairbrot@lancaster.ac.uk
- This information is stored in commit metadata
Initialising and cloning repositories

• An git repository can be initialised with any directory with the following command:
  
git init

• Alternatively, a git repository can be cloned directly with:
  
git clone url
Workflow

1. *Pull* changes from remote repository and merge changes into working tree
2. Make changes to files in project
3. *Stage* changes for local commit
4. Examine changes
5. *Commit* changes to local repository
6. *Push* changes to remote repository
Pulling Changes

• Before working on your files it is a good idea to download any changes from the remote repository, and merge them into your own working tree:

  ```
  git fetch  # Download changes from remote repository
  git merge  # Merge changes into working tree
  ```

• Alternatively, this can be done with the following single command:

  ```
  git pull  # Fetch and merge
  ```
Staging Changes

• After making changes to your working tree, you must *stage* the changes you want to commit:
  
git add file1 file2 ... # Stage files for commit

• Files which need to be moved/renamed or deleted should be done so with the appropriate git commands:
  
git mv
  
git rm

• These commands are used in exactly the same way as the corresponding shell utilities
Examining Changes: Status

- The command `git status` prints a summary of the current state of the working tree:

```
fairbrot@vetinari:~/git/single_airport_model$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   Flexible Scheduler.ipynb
    modified:  Scheduler Comparison.ipynb
    deleted:   setup.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   airport/preferences.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    Redundent flight count.ipynb
    congestion.ipynb
    data/
    hierarchical.ipynb
    schedule_example.ipynb
```
Examining Changes: Diff

- The command `git diff` prints the line-by-line changes made to files since the last commit:

```bash
fairbrot@vetinari:/git/single_airport_model$ git diff
diff --git a/airport/preferences.py b/airport/preferences.py
index 43650a1..11c4e4b 100644
--- a/airport/preferences.py
+++ b/airport/preferences.py
@@ -33,7 +33,7 @@ def peak_interval(t, peak_mat):
     class AbstractDispPreferences(metaclass=ABCMeta):

         def __init__(self, schedule, arr_costs, dep_costs, arr_peaks, dep_peaks,
-            arr_forbidden=None, dep_forbidden=None, tau=None):
+            arr_forbidden=None, dep_forbidden=None, tau=None, alpha=1.0):
            self.T = schedule.T
            self.D = schedule.D
            self.baseline_schedule = schedule
@@ -44,6 +44,7 @@ Class AbstractDispPreferences(metaclass=ABCMeta):
             self.dep_peaks = dep_peaks
             self.arr prefs = dict()
             self.dep prefs = dict()
+            self.alpha = alpha
            self.tau = tau
            if arr_forbidden is not None:
                self.arr_forbidden = arr_forbidden
@@ -205,9 +206,9 @@ class AbstractDispPreferences(metaclass=ABCMeta):

             def airline_disp_cost(self, f, t, arr_or_dep):
                 if arr_or_dep == 'a':
-                    return self.arr_costs[f] * abs(f + t - arr - t)
+                    return self.arr_costs[f] * abs(f - t - arr - t)
```
Examining Changes: Reset

- The `git reset` command can be used to unstage all changes.
- Adding the flag `--hard` will revert all files under version control to their state in the last commit.
Commiting Changes

- To create a new commit, use the `git commit` command:
  ```
git commit -m "Log message"
  ```
- The `-m` flag specifies a required log message associated with the commit.
- These messages should be short and informative.
- Previous commits can be examined through the `git log` and `git reflog` commands:

  ```
<table>
<thead>
<tr>
<th>COMMENT</th>
<th>DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATED MAIN LOOP &amp; TIMING CONTROL</td>
<td>14 HOURS AGO</td>
</tr>
<tr>
<td>ENABLED CONFIG FILE PARSING</td>
<td>9 HOURS AGO</td>
</tr>
<tr>
<td>MISC BUGFIXES</td>
<td>5 HOURS AGO</td>
</tr>
<tr>
<td>CODE ADDITIONS/EDITS</td>
<td>4 HOURS AGO</td>
</tr>
<tr>
<td>MORE CODE</td>
<td>4 HOURS AGO</td>
</tr>
<tr>
<td>HERE HAVE CODE</td>
<td>4 HOURS AGO</td>
</tr>
<tr>
<td>AAAAAAAA</td>
<td>3 HOURS AGO</td>
</tr>
<tr>
<td>ADKFSJLKDFFFFS</td>
<td>3 HOURS AGO</td>
</tr>
<tr>
<td>MY HANDS ARE TYPING WORDS</td>
<td>2 HOURS AGO</td>
</tr>
<tr>
<td>HAAAAAAANDS</td>
<td>2 HOURS AGO</td>
</tr>
</tbody>
</table>
  ```

  As a project drags on, my Git commit messages get less and less informative.
Pushing Changes

- Commits can be added to the remote repository with the following command:
  ```
git push
  ```
- This command will only work if your local repository is synchronised with the remote repository (run `git pull` beforehand)
Demonstrations and Tips
Tips

- Avoid putting files under version control which are produced by other files (e.g. pdfs and executables)
- Where non-text files (e.g. pictures) are required, avoid changing as changes cannot be stored efficiently
- Create a file .gitignore in your working tree to tell git which files to ignore:

```plaintext
*.pdf
*.o
~
*pyc
```

- Avoid using git add .
Help and Links

• Git:
  • Tutorial: https://www.atlassian.com/git/tutorials
  • Run `git help` or `git commit help` in Terminal
• Magit (emacs interface): https://magit.vc/
• SmartGit (GUI): https://www.syntevo.com/smartgit/
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.