UPskill

Digital Skills & Jobs

Git – Basic Commands

Metodologias de Trabalho em Equipa

P.PORTO

apdc
ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITÉCNICOS

Creating a repository

- You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository
2. You can clone an existing Git repository from elsewhere.

Creating a repository

- Access the project folder or create a new one;

Linux
```
$ cd /home/{user}/my_project
```

Windows
```
$ cd C:/Users/{user}/my_project
```

- Initialize the repository

```
$ git init
```

- This command create a .git folder in the project folder
  - *.git* folder is where Git will store and manage the version control history of the project.

Checking the status of your files

- The main tool you use to determine which files are in which state is the `git status` command.

```
$ git status
On branch master
Your branch is up-to-date with
'origin/master'.
nothing to commit, working tree clean
```

  - This means you have a clean working directory (none of your tracked files are modified). Git also doesn't see any untracked files, or they would be listed here. Finally, the command tells you which branch you're on.

Tracking new files

- To begin tracking a new file, you use the command `git add`.
  - Create a new file (file.js) in your file directory.

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be
committed)

    file.js

nothing added to commit but untracked files present (use "git
add" to track)
```

tracking file

```
$ git add file.js
```

Tracking new files

```
$ git status
On branch master
Your branch is up-to-date with
'origin/master'.
Changes to be committed:
  (use "git restore --staged
<file>..." to unstage)

    new file:   file.js
```

Tracking new files

- To add all files to the index

```
$ git add *
```

Staging Modified Files

- Committing changes to the repository is the key step of version control.
- This is where we save a snapshot of the current state of all tracked files.
- To commit our current changes type:

```
$ git commit –m "descriptive message"
```

  - Provide a short, direct, and descriptive commit message

Staging Modified Files

- Edit the file *file.js*
- Check the git status

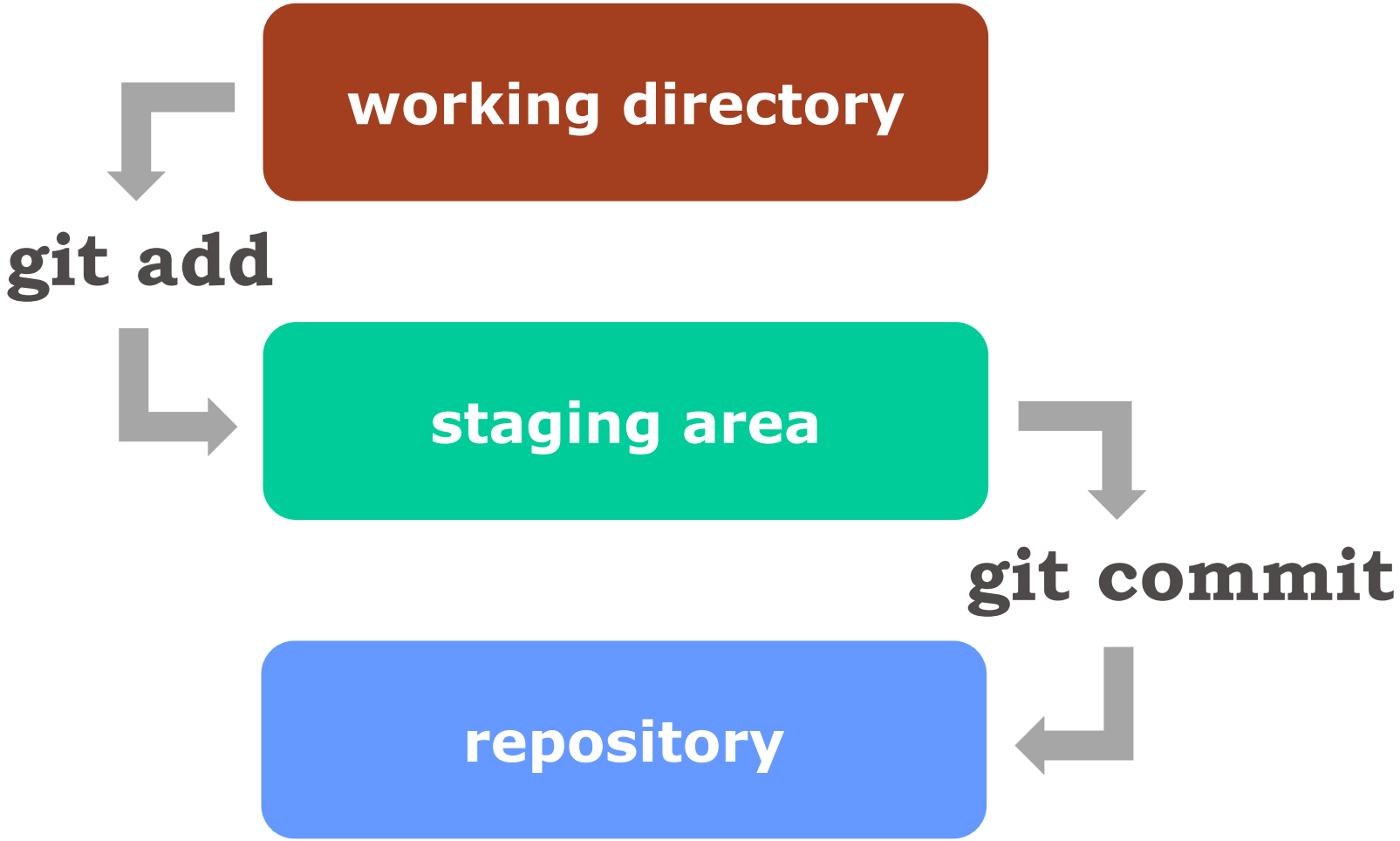
```
$ git status

 On branch master
 Your branch is up to date with 'origin/master'.

 Changes to be committed:
  (use "git restore –staged <file> …" to unstage)
        new file: file.js

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

Staging modified files

- What?? Git now tells us is that *file.js* falls under the category of "Changes not staged for commit".
  - This means the file has changed since the last commit however, we haven't told Git that we want to include these new changes in our next commit.
- To do that, we must "stage" the file using:

```
$ git add file.js
```

Staging Modified Files



Mistakes happen

- If you make a typo in your commit message, or you forget to stage an important change before committing, you can easily amend your last commit using:

```
$ git commit --amend
```

- Example:

```
$ git add file.js
$ git commit –m "Domain class file
aded"
$ git commit --amend
$ git add fname.js
$ git commit –m "Domain classes file
and fname added"
```

typo

forgot to stage

Deleting and moving files

- To delete file.js from repository use:

```
$ git rm file.js
```

  - This will both delete the file from the file system and stage this deletion action for your next commit.

Deleting and moving files

- To stop tracking *file.js* (remove from repository) without deleting it from the file system use:

```
$ git rm –cached file.js
```

- To move or rename use:

```
$ git mv <source> <destination>
```

Commit history

- To display the commit history use `git log`:

```
$ git log

commit 5c9d9f3a8c997d006c197609a4355b979e784d53 (HEAD ->
master)
Author: Paulo Proença <prp@upskil.pt>
Date:   Sat Nov 7 11:35:39 2020 +0000

    Domain classes file and fname added

commit 1d2f05f80b4291a68c4c6b53ca57e87006f8f987
Author: Paulo Proença < prp@upskil.pt >
Date:   Sat Nov 7 10:55:16 2020 +0000

    Initial commit
```

The commit history

- There are a whole of host of flags to change the information and how it appears.

```
$ git log --pretty=format:"%h %s <%an>" --
graph

* 5c9d9f3 Domain classes file and fname added
<Paulo Proenca>
* 1d2f05f Inicial commit <Paulo Proenca>
```

The commit history

- To list only commits of a certain author:

```
$ git log --author="Paulo Proenca"
```

- To investigate all the different options use:

```
$ git help log
```

History of a single file

- Another useful way to visualize the history to is to look at a single file and see in which commit each line was last changed with `git blame`.

```
$ git blame file.js

^1d2f05f (Paulo Proenca     2020-11-07 10:55:16 +0000 1)
var express    = require('express');
5c9d9f3a (Paulo Proenca     2020-11-07 11:35:39 +0000 2)
var bodyParser = require('body-parser');
00000000 (Not Committed Yet 2020-11-07 13:46:49 +0000 3)
d=4;
```

Git Status

- While the `git status` output is pretty comprehensive, it's also quite wordy. Git also has a short status flag so you can see your changes in a more compact way.

```
$ git status -s

 M README
MM Rakefile
A  lib/git.rb
M  lib/simplegit.rb
?? LICENSE.txt
```

Git Status

- New files that aren't tracked have a ?? next to them
- New files that have been added to the staging area have an A
- Modified files have an M
- and so on …

  - the left-hand column indicates the status of the staging area and the right-hand column indicates the status of the working tree.

Git Status

```
$ git status -s

 M README
MM Rakefile
A  lib/git.rb
M  lib/simplegit.rb
?? LICENSE.txt
```

- The README file is modified in the working directory but not yet staged
- lib/simplegit.rb file is modified and staged
- The Rakefile was modified, staged and then modified again, so there are changes to it that are both staged and unstaged.

Stashing

- Git offers a useful feature for those times when your changes are in an incomplete state, you are not ready to commit them, and you need to temporarily return to the last committed.
- This feature is named stash and pushes all your uncommitted changes onto a stack.

```
$ git stash
```

Comparing commits

- To compare commits to see how things have changed use `git diff` command.
- A patch-style view of the difference between the currently edited and committed files (HEAD), or any two points in the past can easily be summoned.
- The .. operator signifies a range is being provided.

```
$ git diff 1d2f05f..5c9d9f3a
```

- An omitted second element in the range implies HEAD as destination

Comparing commits

- This command compares last unstaged changes with the commit referenced as argument.

```
$ git diff <commit hash>
```

- If you don't supply a commit hash it use the last commit.

```
$ git diff
```

- If you supply two commits hashes, the differences between those commits will be displayed

```
$ git diff 1d2f05f..5c9d9f3a
```

Comparing commits

```
$ git diff 5c9d9f3

diff --git a/file.js b/file.js
index 8ae5425..ad7e0a0 100644
--- a/file.js
+++ b/file.js
@@ -1,3 +1,3 @@
 var express    = require('express');
-a=1
\ No newline at end of file
+b=0;
\ No newline at end of file
diff --git a/fname.js b/fname.js
new file mode 100644
index 0000000..fd3c1fb
--- /dev/null
+++ b/fname.js
@@ -0,0 +1 @@
+var express    = require('express');
\ No newline at end of file
```

Tagging

- Git has the ability to tag specific points in a repository's history as being important.
- Typically, people use this functionality to mark release points (v1.0, v2.0 and so on)
- To list the existing tags just use `git tag` command.

Tagging

- To create a tag in the last commit:

```
$ git tag v1.5
```

- To create a tag in a specific commit:

```
$ git tag v1.0 5c9d9f3
```

Tagging

- To delete a tag on your local repository, you can use `git tag -d <tagname>`:

```
$ git tag -d v1.4-lw
Deleted tag 'v1.4-lw' (was e7d5add)
```

  - Note that this does not remove the tag from any remote servers. To achieve this the `$ git push origin --delete <tagname>` must be used.

Treeish & Hashes

- Git marks each commit with a SHA-1 hash that is unique to the committer.
- A full SHA-1 hash has 40 hex characters

```
5c9d9f3a8c997d006c197609a4355b979e784d53
```

- However, the first 7 characters are sufficient to uniquely identify the commit.

```
5c9d9f3
```

Treeish & Hashes

- To efficiently navigate the commit history, several symbolic shorthand notation can be used

Treeish	Definition
Head	The current committed version
Head^	One commit ago
Head^^	Two commits ago
Head~1	One commit ago
Head~3	Three commits ago
:"Reformatting all"	Nearest commit whose message begins with “Reformatting all”
RELEASE-1.0	User-defined tag applied to the code when it was certified for release

Treeish & Hashes

- Treeish can be used in combination with all git commands that accept a specific commit or range of commits.

```
$ git log HEAD~3 HEAD
$ git checkout HEAD^^
$ git merge RELEASE-1.0
```

UPskill

Digital Skills & Jobs

Git – Basic Commands

Metodologias de Trabalho em Equipa

P.PORTO

apdc
ASSOCIAÇÃO PORTUGUESA PARA O DESENVOLVIMENTO DAS COMUNICAÇÕES

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

CONSELHO COORDENADOR DOS INSTITUTOS SUPERIORES POLITÉCNICOS