ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ

CS255 - Programming Lab

Ενότητα: Tutorials

Άγγελος Μπίλας

Τμήμα Επιστήμης Υπολογιστών

Tutorial 5 – Git

Git is a distributed version control system. Version control systems allow developers to track the development of their process and be able to revert changes or go back in time.

- ## Terminology

Repositories

With git there is no need for a single central repository.

Local

Users can create local repositories with git init. Additionally using git clone users may create local clones (copies) of remote repositories.

Remote

Remote repositories are *special* clones that are hosted in a remote server or even in a different directory on the filesystem.

The idea is to have multiple local clones where users keep track of their own work and when it gets to a functional stage they *push*it to the remote repository so that other users can view/get it. Note however that there is no restriction on the number of remote repositories one can use. One might want to use various remote repositories for redundancy.

Workspace

Workspace is where a user acts, it is essentially the local directory where a repository was cloned.

Index

Index, as it names denotes, is the mechanism that keeps track of the (local) changes. When a user changes something in his workspace he needs to *commit* it to update the index.

Stage area

Something between the workspace and the index. It is more like a temporary state that reflects what the user is planning to put in the index.

Unstaged changes

Changes (i.e. modifications, additions, deletions of files) in the workspace that have not been added for commission to the index.

Staged files

Files that are modified or new in the workspace and have been added for commission to the index.

Stash

Stash is a special place where users may temporarily push changes that they don't want to add to the index. They may later pop those changes. It is essentially a buffer holding various changes that for some reason the user does not want to index.

- ## Commands

Creating a repository

git clone path_or_link/to/git/repository

Clones a remote repository. e.g. git clone https://github.com/zakkak/rendezvous.git

git init

Creates a new empty (local) repository in the current directory.

Handling local state (the workspace)

git status

Shows the status of the workspace. It lists any unstaged and staged changes.

git diff

Shows the changes to the tracked files. Tracked files are these files that have been staged at least once in the lifetime of the repository.

git add file

Adds file to the stage area.

```
git add .
```

Stages all current changes.

```
git add -p file
```

Starts staging changes in interactive mode and asks the user which changes to stage.

```
git rm file
```

Removes file from the workspace and stages its removal.

```
git commit
```

Commits any staged changes to the index.

```
git commit -a
```

Commits all changes to the index. It is essentially a shortcut for git add . && git commit.

```
git commit -m "My message"
```

Commits any staged changes to the index and adds the commit message "My message".

```
git commit --amend
```

It allows users to modify the commit message or add some extra changes to the last commit. Note that it is not safe to amend commits that have been already pushed to a remote repository.

History

```
git log
```

Presents the commit logs.

```
git log -p <file>
```

Presents the changes to file by each commit in the history.

```
git log --graph
```

Presents commit logs with more than one branches in a visually appealing manner.

```
git blame <file>
```

Shows which user changed last each line of the file.

Branching

Branches are a very good way to have different versions of the same project in a repository. For example if you want to try a new feature you can create an new branch and implement that feature in it. If you are happy with your new feature you can then merge that branch to the master branch.

```
git branch -a
```

Lists available branches

git checkout <branch>

Changes the the working branch in the workspace.

git branch <new branch>

Creates a new branch from the current index.

git checkout <old branch> -b <new branch>

Creates a new branch from an old one.

git branch -d <branch>

Deletes a local branch.

Remote management

git remote

Lists all available remotes

git remote show <name>

Lists details about the remote *name*.

git remote add <name> <url>

Adds a new remote named *name* with url *url*.

git remote remove <name>

Removes remote *name*.

git push

Pushes (i.e. uploads) any updates of the index to the default remote repository

git push <remote> <branch>

Pushes (i.e. uploads) any updates of the index to the *branch* of *remote*.

git fetch

Fetches (i.e. downloads) any updates of the index from the default remote repository.

git fetch <remote> <branch>

Fetches (i.e. downloads) any updates of the index from *branch* of *remote*.

git pull

Pulls (i.e. downloads and merges) any updates of the index from a remote repository.

git pull <remote> <branch>

Pulls (i.e. downloads and merges) any updates of the index from *branch* of *remote*.

git branch -dr <remote/branch>

Deletes a *branch* from *remote*.

Tags

git tag <name>

Creates tag *name*.

git tag <name> -m "My message"

Creates tag *name* and attached message "My message" to it.

git tag

Lists all available tags.

git tag -n

Lists all available tags along with the attached messages.

git push --tags

Push tags to the default remote repository.

git merge

git merge <branch>

Merges *branch* to the current index.

git merge --abort

Aborts merge.

git add <resolved>

Stages *resolved* and changes its state from conflicting.

git rm <resolved>

Stages *resolved* for removal and changes its state from conflicting.

git mergetool

Use a merge tool to resolve conflicts.

Undo/Revert

git checkout <file>

Drops any changes of *file* and restores it to the last staged state.

git checkout HEAD <file>

Drops any changes of *file* and restores it to the HEAD state in the index.

git revert <commit>

Reverts/undos the changes performed by *commit*. Creates a new commit that reflects this revert.

git reset HEAD

Unstages any changes but preserves them in the workspace.

git reset <commit>

Moves the index HEAD to *commit* preserving any changes since that commit in the workspace as unstaged.

git reset --hard HEAD

Drops all changes and restores the workspace to the HEAD state in the index.

git reset --hard <commit>

Drops all changes and restores the workspace to *commit* and moves the index HEAD to it. Should not be used to move HEAD before pushed commits, in that case you should use git revert.

Online interactive tutorial

There is also available an interactive tutorial here

Authored by: Foivos S. Zakkak

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Κρήτης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ
2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης