

# Διάλεξη 2 - Συναρτήσεις και Version Control

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός

# Ανακοινώσεις / Γενικά

- 16 Οκτωβρίου δεν θα έχουμε διάλεξη και εργαστήρια
- 18 Οκτωβρίου θα έχουμε αναπλήρωση 7μμ-9μμ στο Αμφιθέατρο
- Βγήκε η Εργασία #0 - Προθεσμία 8 Νοεμβρίου, 23:59
- Ερωτήσεις / Απαντήσεις / Διευκρινήσεις
  1. Windows 11 -> έχει ssh built in (δεν χρειάζεται putty)
  2. Σύνδεση Wifi -> βεβαιωθείτε ότι έχετε certificates
  3. Δεν έχω gcc τι κάνω; apt (θα το καλύψουμε σήμερα)
  4. Τυφλό σύστημα (touch typing) - βρείτε ένα site και μάθετε!
  5. Macbooks: μπορώ να κάνω compile - ναι, αλλά ο έλεγχος της εργασίας σε Linux
  6. Συγγράμματα διαθέσιμα
  7. 2ο Φυλλάδιο εργαστηρίου διαθέσιμο στο [progintro.github.io](https://progintro.github.io)
  8. Δεν αλλάζουμε εργαστήριο χωρίς να συνεννοηθούμε με τον υπεύθυνο

# Την Προηγούμενη Φορά

- Γραμμή Εντολών
- Μεταγλώττιση
- Γράψαμε και τρέξαμε ένα "Hello World" Linux C Πρόγραμμα

# Διάλεξη 2 - Συναρτήσεις και Version Control

Σήμερα:

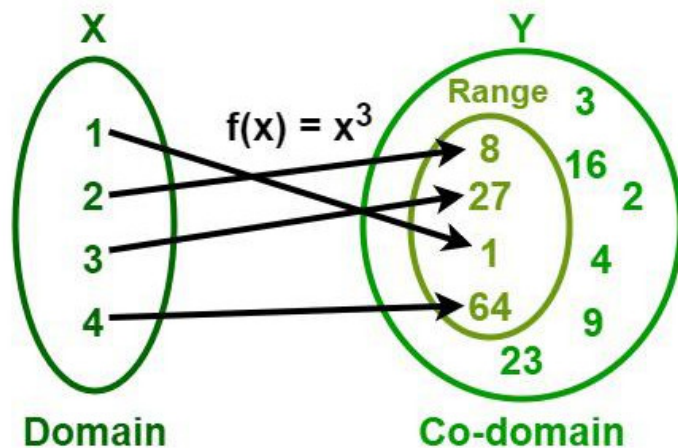
1. Συναρτήσεις και Ακέραιοι (Μαθηματικά -> C)
2. Πρόγραμμα για Υπολογισμό Βαθμολογίας
3. Παραγοντικό
4. Version Control
5. Git

# Συναρτήσεις

Συνάρτηση: Μια αντιστοίχιση μεταξύ δύο συνόλων, που καλούνται σύνολο ορισμού και σύνολο τιμών, κατά την οποία κάθε ένα στοιχείο του πεδίου ορισμού αντιστοιχίζεται σε ένα και μόνο στοιχείο του πεδίου τιμών.

$$f(x) = x^3$$

$$f : \mathbb{Z} \rightarrow \mathbb{Z}$$



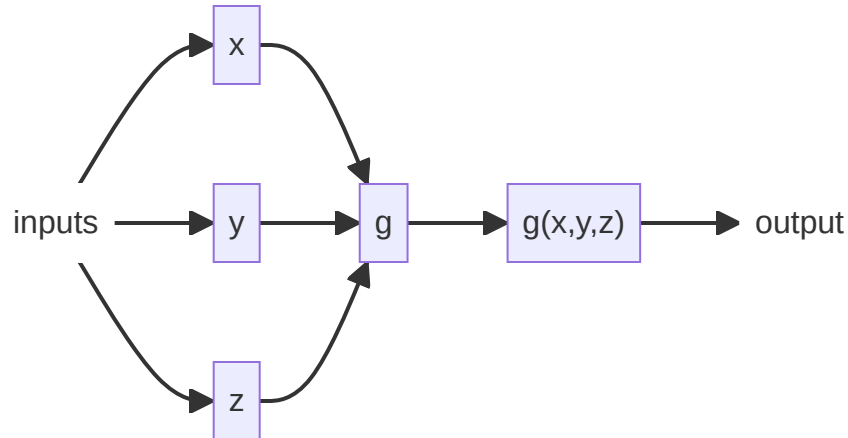
# Πολυπαραμετρικές Συναρτήσεις

$$g(x, y, z) = x^2 + y^2 + z^2 + 42$$

$$g : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$$

$$h(x, y) = x + y + 1$$

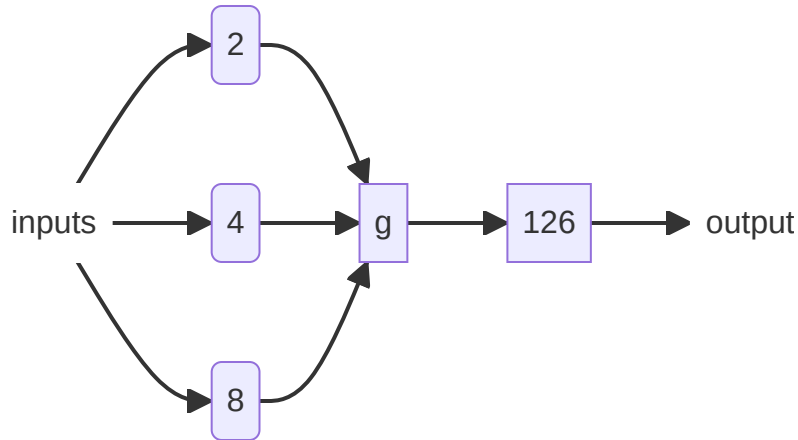
$$h : \mathbb{R} \times \mathbb{N} \rightarrow \mathbb{R}$$



# Αποτίμηση Συναρτήσεων

$$g(x, y, z) = x^2 + y^2 + z^2 + 42$$

$$g(2, 4, 8) = 2^2 + 4^2 + 8^2 + 42 = 126$$



# Συναρτήσεις στην C

Συνάρτηση είναι μια σειρά υπολογισμών που παίρνουν τις εισόδους της συνάρτησης και παράγουν την έξοδο.

Όπως και στα μαθηματικά, κάθε δεδομένο εισόδου και εξόδου της συνάρτησης έχει έναν **τύπο**, το σύνολο τιμών που μπορεί να πάρει. Για παράδειγμα, ο τύπος `int` στην C αντιπροσωπεύει τους ακεραίους (integers).

```
int g(int x, int y, int z) {  
    return x * x + y * y + z * z + 42;  
}
```

Ο ορισμός μιας συνάρτησης αποτελείται από 3 μέρη:

1. Το *όνομα της συνάρτησης* και τον *τύπο της τιμής που επιστρέφει* στην μορφή `τύπος όνομα`.
  - `int g` στο παράδειγμα. Αν καλέσουμε `g(...)` περιμένουμε ακέραιο αποτέλεσμα.
2. Τα *δεδομένα εισόδου ή ορίσματα* της συνάρτησης εντός παρενθέσεων, επίσης της μορφής `τύπος όνομα`.
  - `int x`, `int y`, `int z` είναι 3 ακέραια ορίσματα με ονόματα `x`, `y` και `z`.
3. Το *σώμα της συνάρτησης* εντός των αγκυλών `{ }` περιέχει τον υπολογισμό της τιμής της



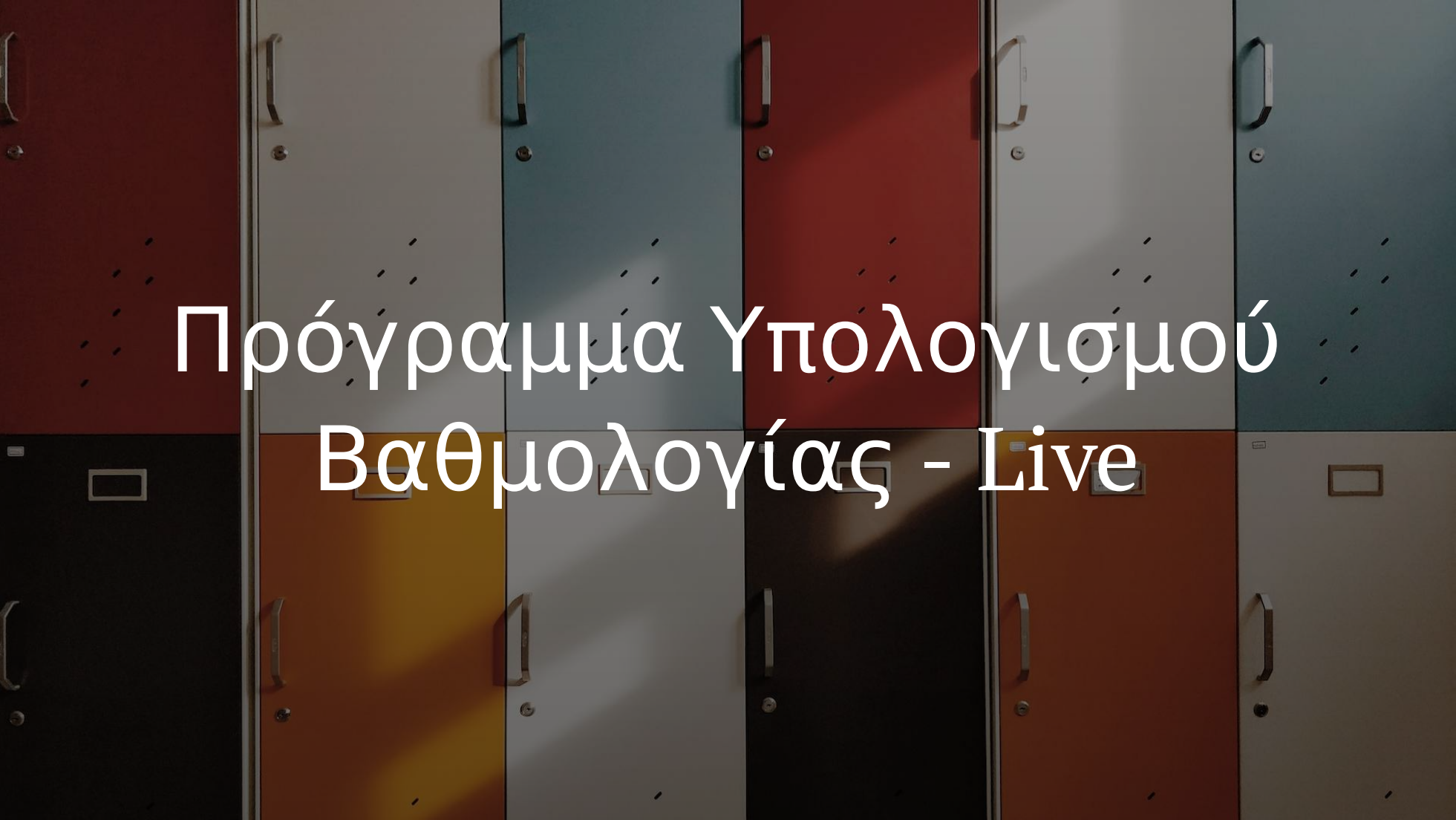
# Υπολογισμός Βαθμολογίας Πρωτοετών

- $50\% * \text{Τελική Εξέταση} + 30\% * \text{Ασκήσεις} + 20\% * \text{Εργαστήριο}$
- Έστω ότι χρησιμοποιούμε 3 ακεραίους για να αναπαραστήσουμε τα αποτελέσματα [0 - 100].
  - `int final_exam`
  - `int homework`
  - `int lab`

- Στα μαθηματικά:

$$\text{grade}(\text{final\_exam}, \text{homework}, \text{lab}) = \text{final\_exam} \times 50\% + \text{homework} \times 30\% + \text{lab} \times 20\%$$

- Έλεγχος ορθότητας: `grade(70, 80, 100) == 79?`
- Σε C;



# Πρόγραμμα Υπολογισμού Βαθμολογίας - Live

# Χρειαζομαι εθελοντή / εθελόντρια

ssh ubuntu@44.202.155.226

pass:

# Υλοποίηση Υπολογισμού Βαθμολογίας 1/2

```
#include <stdio.h>
#include <stdlib.h>

// Compute grades using the class formula
int grade(int final_exam, int homework, int lab) {
    return final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100;
}

int main(int argc, char **argv) {
    if (argc != 4) {
        printf("Program needs to be called as `./prog final_exam homework lab`\n");
        return 1;
    }
    int final_exam = atoi(argv[1]);
    int homework = atoi(argv[2]);
    int lab = atoi(argv[3]);
    printf("Grade: %d\n", grade(final_exam, homework, lab));
    return 0;
}
```

# Υλοποίηση Υπολογισμού Βαθμολογίας 2/2

- Η συνάρτηση `main` έχει μεταβλητό αριθμό ορισμάτων. Η παράμετρος `argc` λέει πόσα ορίσματα περάσαμε στο πρόγραμμα, ενώ η παράμετρος `argv` περιέχει το κείμενο που αντιστοιχεί σε κάθε όρισμα.
  - `argv[1]` περιέχει το 1ο όρισμα του προγράμματος, `argv[2]` το 2ο, κ.ο.κ.
- Η συνάρτηση βιβλιοθήκης (`stdlib.h`) `atoi` μετατρέπει ένα όρισμα από αλφαριθμητικό (ASCII) σε έναν ακέραιο (integer).
- Εάν σας ενοχλεί το πόσο "μαγική" φαίνεται η `main` και τα ορίσματά της, μην εξάπτεστε! Είναι μια από τις δυσκολότερες συναρτήσεις που θα συναντήσουμε στην C και τις επόμενες βδομάδες θα χτίσουμε το υπόβαθρο για να γίνει πιο κατανοητή.

# Υπολογισμός Βαθμολογίας (Γενικός)

- Αν είστε πρωτοετείς:
  - $50\% * \text{Τελική Εξέταση} + 30\% * \text{Ασκήσεις} + 20\% * \text{Εργαστήριο}$
- Αλλιώς:
  - $70\% * \text{Τελική Εξέταση} + 30\% * \text{Ασκήσεις}$
- Έστω ότι έχουμε μια μεταβλητή `int year` που δηλώνει το έτος στο οποίο βρίσκεσαι
- Στα μαθηματικά:

$grade(final\_exam, homework, lab, year) =$

$$\begin{cases} final\_exam \times 50\% + homework \times 30\% + lab \times 20\% & , year \leq 1 \\ final\_exam \times 70\% + homework \times 30\% & , year > 1 \end{cases}$$

- Σε C;

# Υπολογισμός Βαθμολογίας "Τελικό" - Είναι Σωστό;

```
#include <stdio.h>
#include <stdlib.h>

// Compute grades using the class formula
int grade(int final_exam, int homework, int lab, int year) {
    if (year <= 1) {
        return final_exam * 50 / 100 + homework * 30 / 100 + lab * 20 / 100;
    } else {
        return final_exam * 70 / 100 + homework * 30 / 100;
    }
}

int main(int argc, char **argv) {
    if (argc != 5) {
        printf("Program needs to be called as `./prog final_exam homework lab year`\n");
        return 1;
    }
    int final_exam = atoi(argv[1]);
    int homework = atoi(argv[2]);
    int lab = atoi(argv[3]);
    int year = atoi(argv[4]);
    printf("Grade: %d\n", grade(final_exam, homework, lab, year));
    return 0;
}
```

# Η Συνάρτηση Παραγοντικό (Factorial)

- Στα μαθηματικά το παραγοντικό ενός φυσικού αριθμού  $n$ , συμβολίζεται με  $n!$  και είναι το γινόμενο όλων των θετικών ακεραίων μικρότερων ή ίσων του  $n$ .

Για παράδειγμα:

$$2! = 1 \cdot 2 = 2$$

$$3! = 1 \cdot 2 \cdot 3 = 6$$

$$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$$

$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = ?$$

$$10! = 1 \cdot 2 \cdot 3 \cdots 9 \cdot 10 = ?$$



# Η Συνάρτηση Παραγοντικό (Factorial)

- Στα μαθηματικά το παραγοντικό ενός φυσικού αριθμού  $n$ , συμβολίζεται με  $n!$  και είναι το γινόμενο όλων των θετικών ακεραίων μικρότερων ή ίσων του  $n$ .

Ο ορισμός του παραγοντικού στα μαθηματικά είναι αναδρομικός (recursive):

$$n! = \begin{cases} 1 & , n \leq 1 \\ n * (n - 1)! & , n > 1 \end{cases}$$

- Σε C;

# Υπολογισμός Παραγοντικού

```
#include <stdio.h>
#include <stdlib.h>

// Compute the factorial of a number using the recursive
// formula.
int factorial(int number) {
    if (number == 0) {
        return 1;
    } else {
        return number * factorial(number - 1);
    }
}

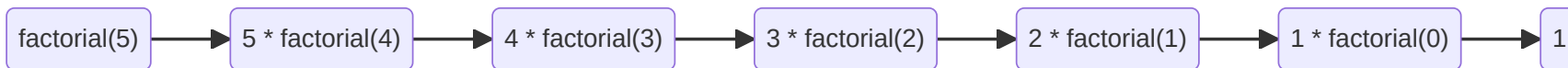
int main(int argc, char **argv) {
    if (argc != 2) {
        printf("Program needs to be called as `./prog number`\n");
        return 1;
    }
    int number = atoi(argv[1]);
    printf("%d! = %d\n", number, factorial(number));
    return 0;
}
```

# Παρατηρήσεις

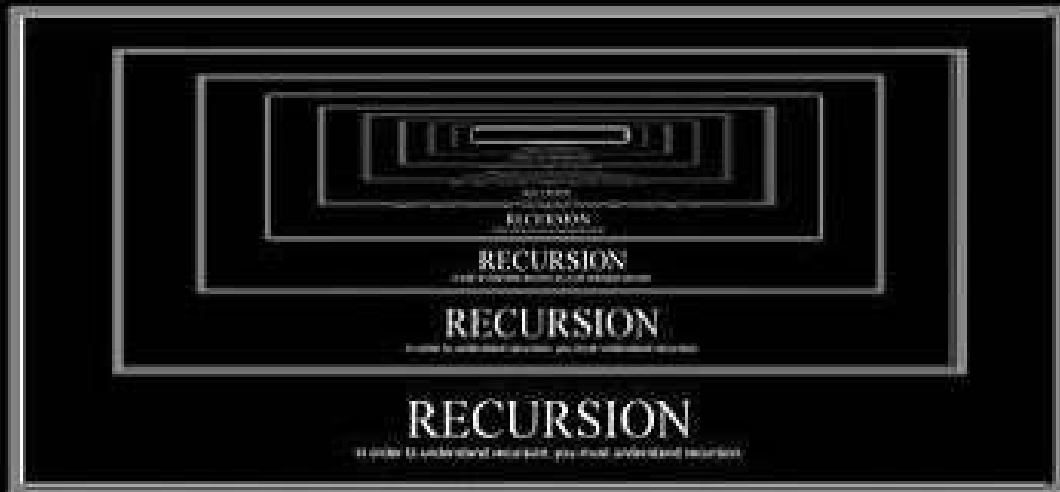
- Για κάποιους αριθμούς το αποτέλεσμα είναι αρνητικό. Τι συμβαίνει; (στο επόμενο μάθημα!)

```
$ ./fact 20  
20! = -2102132736
```

- Πόσες αναδρομικές (στον εαυτό της) κλήσεις κάνει η συνάρτηση `factorial(5)`;



- Πόσες αναδρομικές (στον εαυτό της) κλήσεις κάνει η συνάρτηση `factorial(N)`;
- Τι θα συμβεί αν δώσουμε έναν αρνητικό αριθμό στην συνάρτησή μας; Π.χ., το -42;



## RECURSION

In order to understand recursion, you must understand recursion.

# RECURSION

In order to understand recursion, you must understand recursion.

# RECURSION

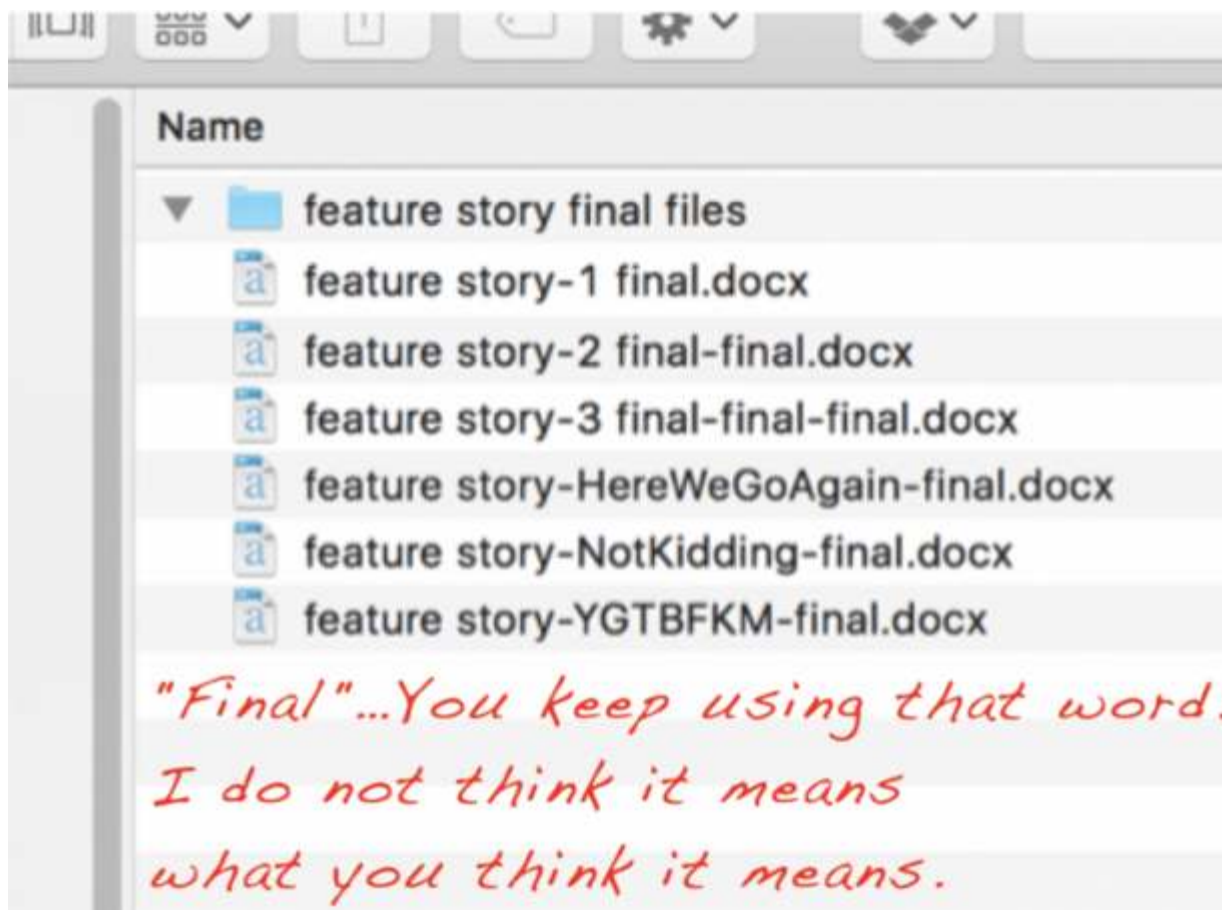


# Version Control και Git

# Version Control - Έλεγχος Έκδοσης

- Ένα σύστημα που επιτρέπει να παρακολουθούμε όλες τις αλλαγές που γίνονται στον κώδικα και να κρατάμε ιστορικό αρχείο.

Γιατί να θέλουμε κάτι τέτοιο;;



# Version Control - Έλεγχος Έκδοσης

- Ένα σύστημα που επιτρέπει να παρακολουθούμε όλες τις αλλαγές που γίνονται στον κώδικα και να κρατάμε ιστορικό αρχείο.

## Γιατί να θέλουμε κάτι τέτοιο;;

- Τα περισσότερα προγράμματα αλλάζουν με τον χρόνο - δεν υπάρχει "τελική μορφή"
- Κάνουμε αλλαγές και κρατάμε μόνο τα αρχεία που χρειαζόμαστε χωρίς να χάνουμε πληροφορία
- Μπορούμε να ανατρέξουμε σε οποιαδήποτε αλλαγή έγινε ανά πάσα στιγμή



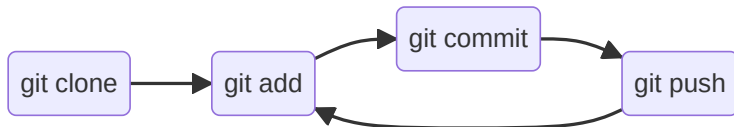
# Git

---

- Είναι το δημοφιλέστερο πρόγραμμα Version Control σήμερα (~90% των χρηστών/project)
- Πρώτη έκδοση γράφτηκε από τον Linus Torvalds το 2005
- Μια δημοφιλής και δωρεάν (για την ώρα) πλατφόρμα στην οποία μπορούμε να αποθηκεύσουμε τον κώδικά μας είναι το <https://github.com/>
  - Θα το χρησιμοποιήσουμε για την γραφή και υποβολή των εργασιών μας

# Git Development Cycle

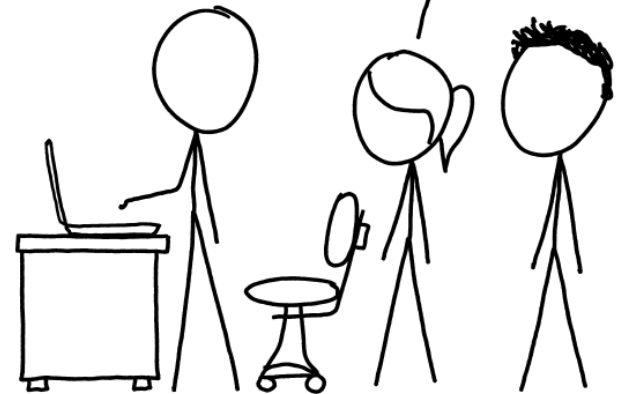
Σε επίπεδο βασικής χρήσης (αυτό που χρειαζόμαστε για το μάθημα), η γραφή κώδικα και η αποθήκευσή με git περιλαμβάνει 4 βήματα:



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.



# Clone repository

`git clone`: Αντιγράφει το repository (αποθετήριο) με όλα τα αρχεία σε έναν τοπικό φάκελο.

```
git clone git@github.com:progintro/hw0-barbouni-2005.git
```

- Αν ελέγξουμε τον φάκελο παρατηρούμε ότι έχει ένα README.md αρχείο.
- Τρέχοντας `git status` μέσα στον φάκελο μπορούμε να δούμε αν υπάρχουν καθόλου αλλαγές.
- Η παραπάνω εντολή θα δημιουργήσει έναν τοπικό φάκελο (αν δεν υπάρχει) με το όνομα hw0-barbouni-2005
- Τα βήματα που ακολουθούμε εδώ, προϋποθέτουν ότι έχετε φτιάξει ένα κλειδί για τον Github λογαριασμό σας και έχετε τελειώσει configuration όπως περιγράφεται στο παράρτημα του Φυλλαδίου 2 του εργαστηρίου.

# Προσθήκη Αρχείου

`git add`: Προσθέτει τα αρχεία στο staging area (προσωρινή λίστα) για μελλοντική αποθήκευση στο repository.

- Δημιούργησε ένα καινούριο αρχείο, π.χ., `touch command/solution.txt`.
- Τρέξε `git status` - θα παρατηρήσεις ότι το καινούριο αρχείο καταγράφεται ως untracked.
- Για να το προσθέσουμε: `git add command/solution.txt`
- Τρέξε `git status` ξανά - θα παρατηρήσεις ότι το αρχείο είναι έτοιμο να γίνει commit (καταχωρηθεί).

# Καταχώρηση Αλλαγών

`git commit`: Αποθηκεύει τις αλλαγές στο τοπικό repository.

- Τρέξε `git commit` και δώσε ένα μήνυμα που περιγράφει την αλλαγή σου.
- Τρέξε `git status` και πάλι - τώρα το μόνο που μένει είναι να σώσουμε τις αλλαγές μας και εκτός του υπολογιστή μας.

# Ανέβασμα Αλλαγών

`git push`: Ανεβάζει τις τοπικές αλλαγές στον απομακρυσμένο server (εξυπηρετητή).

- Έλεγξε ότι οι αλλαγές σου εμφανίζονται στο Github!

# Κατέβασμα Αλλαγών

`git pull`: Κατεβάζει αλλαγές από τον απομακρυσμένο server (αν υπάρχουν).

- Χρήσιμο όταν έχεις πολλούς προγραμματιστές ή πολλούς υπολογιστές και δεν θέλεις να κάνεις clone κάθε φορά.
- Κάνε κάποιες αλλαγές στο repository από το GitHub UI και μετά τρέξε `git pull`.

git commit



git push



git add .



# Για την επόμενη φορά

- Ολοκληρώστε τις εγγραφές στα εργαλεία του μαθήματος.
- Ξεκινήστε να ασχολείστε με την Εργασία #0.
- Προαιρετικό: Git Cheat Sheet
- Προαιρετικό: Git Tutorial.
- Από τις σημειώσεις του κ. Σταματόπουλου, συνιστώ να διαβάσετε τις σελίδες 30-35, 58-71.



Ευχαριστώ και καλή μέρα εύχομαι!

Keep coding!