



Laboratorul 01

Introducere în Programarea Android

Proiectarea și Dezvoltarea Serviciilor Distribuite (PDSD)
Semestrul de Primăvară 2015

Departamentul de Calculatoare



Conținut

- Sistemul de Operare Android**
- Cerințe pentru Dezvoltarea unei aplicații Android**
- Sisteme de Control a Versiunilor (GIT)**
- Crearea unei Aplicații Android în Eclipse Luna SR1a (4.4.1)**
- Testarea Exemplelor de Aplicații Android**



Sistemul de Operare Android

- Android – sistem de operare mobil
 - kernel Linux + biblioteci Java
 - idee – transformarea dispozitivelor mobile în mașini de calcul
 - proiect open-source, tendințe de “acaparare” din partea Google
 - abordare unitară pentru rularea aplicațiilor

- Versiuni
 - în prezent, 5.0.2 (lansat pe 19 decembrie 2014)
 - nume de dulciuri (Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop)
 - de urmărit distribuția dispozitivelor de piață ⇒ decizie de implementare folosind un nivel de API sau altul
 - >= API 16: peste 85% din dispozitive



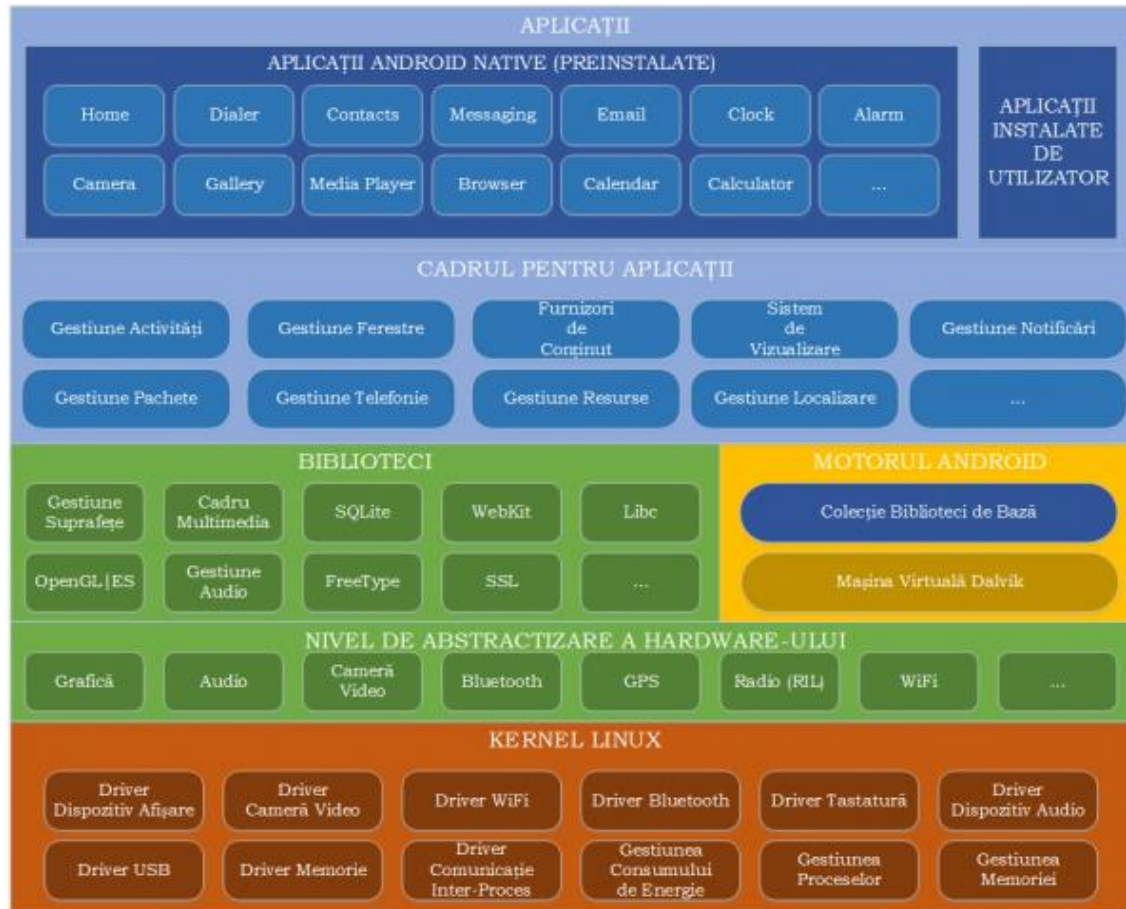
Sistemul de Operare Android (2)

□ Arhitectura – 4 niveluri

1. Kernel Linux – drivere, gestiune memorie, procese
2. Biblioteci (user-space) – SQLite, WebKit, FreeType, SurfaceManager, Bionic, SSL, OpenCORE
3. Motor Android
 - a) biblioteci de bază (Java) – acces la funcțiile telefonului, GUI, furnizori de conținut
 - b) DVM (Dalvik Virtual Machine) – înlocuit de ART (Android Runtime) din 5.0
 - arhitectură bazată pe regiștri
 - compilator JIT
 - bazat pe o implementare Apache Harmony
 - bytecode compilat în fișiere .dex ~ <50% din dimensiunea unui .jar
 - executabilul poate fi modificat la instalare, rulează într-un proces dedicat
4. Cadrul pentru Aplicații – expune funcționalități Android către programatori
5. Aplicații – preinstalate + instalate de utilizator



Sistemul de Operare Android (3)





Sistemul de Operare Android (4)

- Funcționalități
 - stocare – SQLite
 - conectivitate (GSM/CDMA, GPRS, EDGE, 3G, Bluetooth, WiFi, ...)
 - WiFi Direct
 - Android Beam (bazat pe NFC)
 - mesagerie (SMS/MMS)
 - navigare Internet
 - multimedia – numeroase formate
 - grafică 2D/3D
 - senzori: accelerometru, cameră foto, busolă digitală, proximitate, GPS
 - multi: tasking, touch, language
 - GCM – transmitere de notificări
 - partajarea conexiunilor (tethering)



Sistemul de Operare Android (5)

□ Android vs. iPhone

- cote de piață
 - telefoane inteligente: Android ~80%, Apple 15% (grație Iphone 6)
 - tablete – în declin, Apple – lider absolut
- aplicații
 - segment extrem de profitabil, au depășit profitul generat de Hollywood
 - sociale, business
 - ~1.5 mil în Google Play (cu 25% mai multe descărcări), ~1.25 mil în App Store
 - profit din vânzări: Apple >> Google
- oportunități pentru companii
 - Apple – MacOS X, mai puțin răspândit + Objective-C
 - Android – Java (orice platformă) + IDE-uri gratuite

□ Comunitatea programatorilor Android

- Google Android Training
- Android Discuss

Cerințe pentru Dezvoltarea unei Aplicații Android



1. kit de dezvoltare pentru limbajul de programare Java
2. SDK pentru Android
3. mediu integrat de dezvoltare
 - a. Eclipse Luna SR1a (4.4.1) – consum mai mic de memorie, necesită plugin pentru Android
 - b. Android Studio – bazat pe IntelliJ IDEA
4. dispozitiv pentru rulare
 - a. fizic – Dev Tools → Developer Options → Debugging → USB Debugging
 - b. virtual
 - i. Genymotion – viteză mai mare, funcționalități mai multe
 - ii. AVD (livrat cu SDK-ul pentru Android) – poate folosi pentru virtualizare kvm, HAXM



Sisteme de Control al Versiunilor

- funcționalitate
 - revenirea la o versiune anterioară (stabilă) în caz de eroare
 - colaborarea între persoane care lucrează la același proiect

- categorii
 - **locale** – versiunile sunt reținute sub formă de diferențe
 - **centralizate**
 - server central – baza de date cu istoricul modificărilor: single point of failure
 - CVS, Subversion, Perforce
 - **distribuite**
 - istoricul modificărilor – replicat pe toate mașinile, poate fi refăcut cu ușurință; gestiunea depozitelor la distanță
 - Git, Mercurial, Bazaar, Darcs

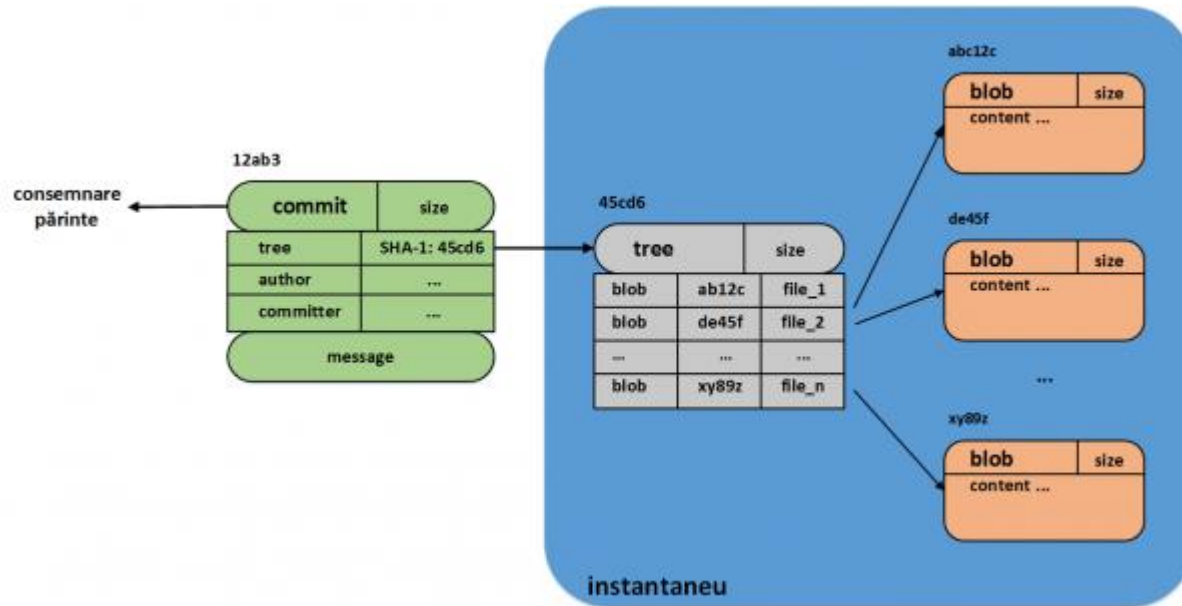


GIT

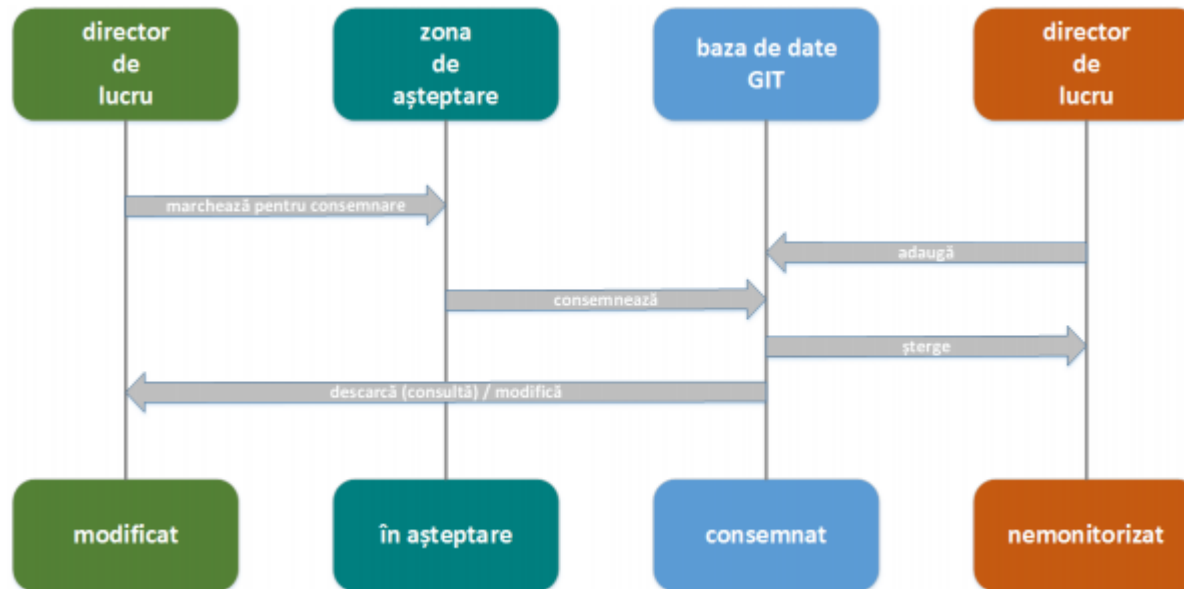
- istoric – strâns legat de dezvoltarea kernelului pentru Linux
 - 1991-2002: arhive de modificări
 - 2002-2005 – BitKeeper, sistem distribuit (devenit între timp proprietar)
 - > 2005 – Git (viteză, arhitectură scalabilă, suport pentru dezvoltare non-liniară, distribuite, capacitate de gestiune a unor proiecte mari)

- caracteristici
 1. snapshot – formă de stocare a unui commit
 - a. tree – referință către fiecare resursă modificată
 - b. blob – resursa propriu-zisă
 2. operații realizate local ⇒ viteză îmbunătățită
 3. integritate – sume de control SHA-1
 4. posibilitate redusă de pierdere a datelor
 5. stările unui fișier
 - a. committed ⇔ GIT directory
 - b. modified ⇔ working directory
 - c. staged ⇔ staging area
 - d. untracked (binare, jurnale) ⇔ .gitignore

GIT (2)



GIT (3)





GIT (4)

□ moduri de lucru

- local

`git init` → `.git` – baza de date GIT în care se stochează istoricul modificărilor

- la distanță

`git clone <URL> [<local_directory>]`

protocoale suportate: `git`, `https`, `ssh`

□ operații

- `git status` – starea fișierelor din working area (untracked, changes to be committed [new file, modified, renamed, deleted], changes not staged for commit)
- `git add <files/directory>` – transfer din working area în staging area
 - `git reset HEAD <file>` - transfer din staging area
 - `git checkout -- <file>` - (periculos !!!) restaurarea unui fișier din working area la starea din baza de date GIT



GIT (5)

□ operații

- `git mv <source> <target>` - redenumirea unui fișier
- `git rm <file/directory>` - ștergerea unui fișier
 - `-f` – fișierul a fost modificat de la commit
 - `-r` (recursiv) – pentru directoare
 - `--cached` – fișiere incluse în commit, care se doresc a fi nemonitorizate

□ ignorarea unor tipuri de fișiere (binare, jurnale) - `.gitignore`

1. câte o regulă pe linie
2. expresii regulate
 - a. `*` - 0 sau mai multe caractere
 - b. `**` - conținutul unui director
 - c. `?` – un singur caracter
 - d. `{ }` – colecții de șabloane
 - e. `[]` – seturi de caractere / intervale (separate prin `-`)
3. `!` (înaintea unui șablon) – negare
4. `#` (la început de linie) - comentariu



GIT (6)

- ❑ vizualizarea modificărilor – `git diff`
 - fără parametri – diferențe între working area și staging area
 - `--cached` / `--staged` – diferențe între staging area și GIT directory

- ❑ consemnarea modificărilor – `git commit`
 - fără parametri – lansare editor (indicat de `core.editor`) pentru indicarea mesajului
 - `-v` – rezultatul comenzii `git diff`
 - `-m` – mesajul propriu-zis
 - `-a` – se omite transferul dintre working area și staging area
 - `--amend` – suprascrierea unei consemnări cu conținutul din zona de așteptare curentă

- ❑ etichete – `git tag [<tag_name>] [<commit_sha-1>]`
 - `lightweighted` – doar suma de control
 - `annotated` – obiecte în directorul GIT + autor / mesaj
 - `-a` – etichete nesemnate
 - `-s` – etichete semnate cu GPG (GNU Privacy Guard)
 - `-m` – mesajul asociat etichetei
 - `-v` – verificarea unei etichete semnate



GIT (7)

- ❑ vizualizarea istoricului de versiuni – `git log`
 - lista de consemnări, în ordine invers cronologică
 - informații: suma de control, autorul, data/ora, mesajul
 - opțiuni
 - `-<n>` - cele mai recente n consemnări
 - `--since`, `--after` – consemnări după o dată/oră absolută/relativă
 - `--until`, `--before` – consemnări înainte de dată/oră absolută/relativă
 - `--author`, `--committer` – consemnările având un anumit autor/contributor
 - `--grep` – consemnări conținând anumite cuvinte în mesajul asociat
 - `--<path>` - consemnări asupra unor fișiere localizate în calea specificată
 - alte opțiuni: `-p`, `--word_diff`, `--stat`, `--short_stat`, `--name_only`, `--name_status`, `--abbrev-commit`, `--relative-date`, `--graph`, `--pretty`, `--oneline`

- ❑ depozite la distanță – încărcare / descărcare cod sursă între mai mulți utilizatori care lucrează la același proiect



GIT (8)

□ depozite la distanță

○ vizualizare referințe către depozite la distanță – `git remote`

- `-v` – afișare URL locație
- `origin` – depozitul la distanță din care s-a realizat operația de clonare

○ adăugarea unei referințe către un depozit la distanță

`git remote add <remote_name> <URL>`

○ descărcarea de cod sursă de pe un depozit la distanță

- `git clone <URL>` - toate branch-urile + monitorizare master
- `git fetch <remote_name>` - descarcă doar actualizările care nu se găsesc pe discul local, fără a realiza integrarea
- `git pull` – descarcă doar actualizările dintr-un branch monitorizat, realizând integrarea

○ încărcarea de cod sursă pe un depozit la distanță

- `git push <remote_name> <branch>`
- trebuie realizată actualizarea directorului de lucru local dacă în depozitul la distanță au fost realizate modificări



GIT (9)

□ depozite la distanță

- consultarea conținutului unui depozit la distanță

```
git remote show <remote-name>
```

- URL
- branch curent
- branch pe care se face `git push`
- branch-uri pe care se face `git pull`
- branch-uri pe server care nu sunt în directorul local, branch-uri în directorul local care nu sunt pe server

- redenumirea unei referințe către un depozit la distanță

```
git remote rename <old_remote_name> <new_remote_name>
```

- ștergerea unei referințe către un depozit la distanță

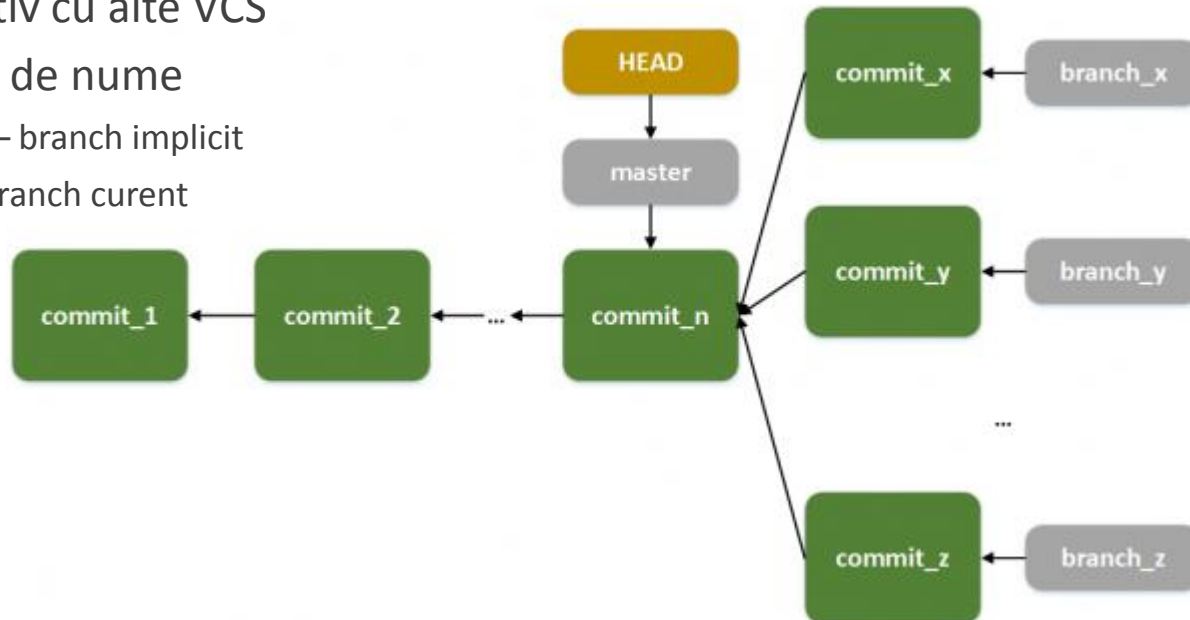
```
git remote rm <remote_name>
```



GIT (10)

gestiune branch-uri

- branch = punct din care dezvoltarea proiectului se realizează non-linear
- versiuni stabile (în exploatare), în lucru, diferite funcționalități
- referință către o consemnare (40 de caractere hexa) ⇒ gestiune foarte rapidă comparativ cu alte VCS
- convenții de nume
 - master – branch implicit
 - HEAD – branch curent





GIT (11)

o operații

1. vizualizare branch-uri – `git branch`
 - * - branch-ul curent
 - -v – se indică cel mai recent commit pentru fiecare branch
 - --merged – doar branch-urile integrate în branch-ul curent
 - --no-merged – doar branch-urile care nu au fost integrate în branch-ul curent (pot fi șterse numai cu opțiunea -D)
2. crearea unui branch – `git branch <branch_name>`
 - inițial, referință către master
 - pe măsură ce se face commit, indică cea mai recentă actualizare
3. transferul între branch-uri – `git checkout <branch_name>`
 - modificarea pointer-ului HEAD + modificarea conținutului directorului de lucru
 - în prealabil, se face commit pentru actualizările din directorul de lucru
4. crearea unui branch + transferul între branch-uri - `git checkout -b <branch_name>`



GIT (12)

o integrarea conținutului a două branch-uri

1. merge

- reflectă fidel istoricul operațiilor
- dificil de gestionat

```
git checkout master
```

```
git merge <branch_name>
```

- fast-forward = pointer-ii master și HEAD sunt mutați înainte

- se identifică cel mai recent strămoș comun și se integrează modificările raportate la acesta

2. rebase

- istoricul operațiilor apare ca liniar, gestiunea referințelor către stările proiectului se face ușor
- !!! de evitat în cazul unor commit-uri pe repository-uri la distanță

```
git checkout <branch_name>
```

```
git rebase master
```

```
git checkout master
```

```
git merge <branch_name>
```

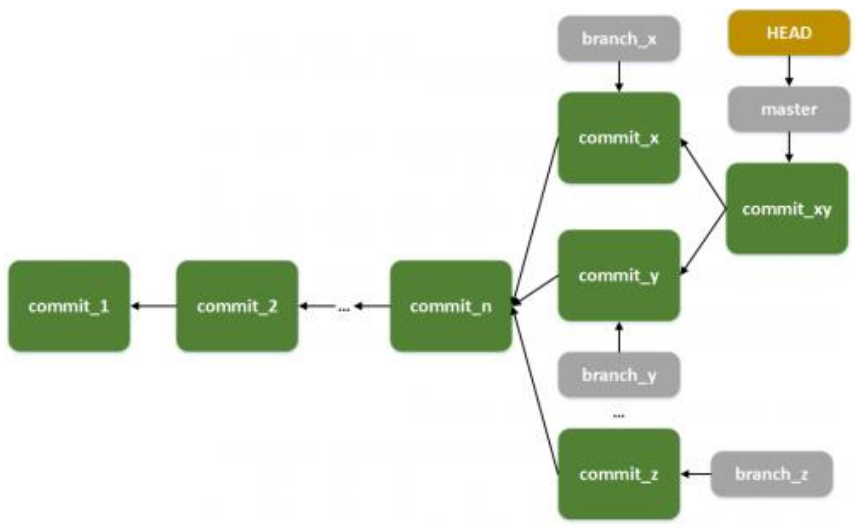
```
- git rebase master <branch_name> =
```

```
git checkout <branch_name> + git rebase master
```

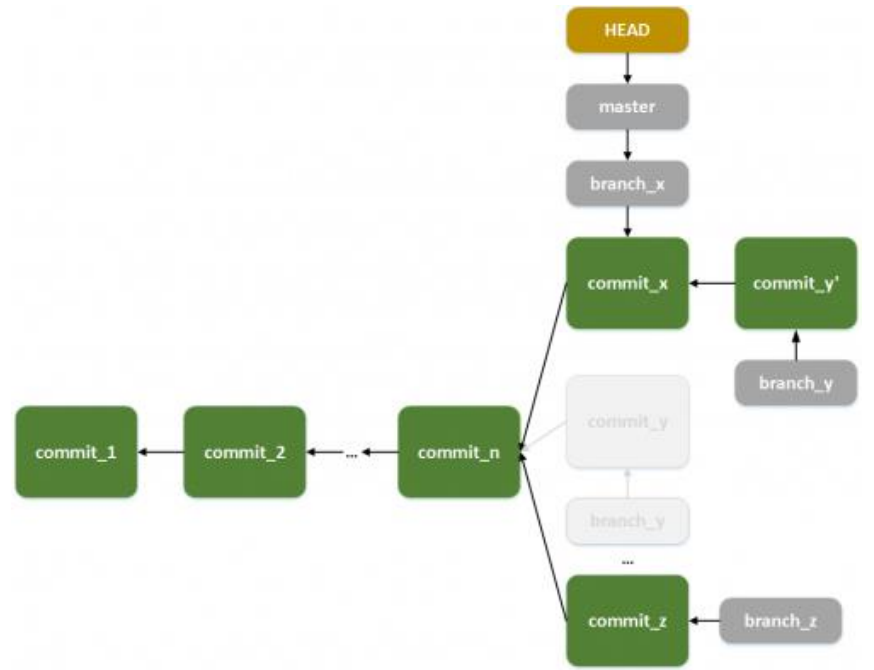
```
- git rebase --onto <branch_to_integrate_in>
```

```
<branch_to_be_omitted> <branch_to_be_included>
```

GIT (13)



MERGE



REBASE



GIT (14)

- ștergerea unui branch
 - `git branch -d <branch_name>`
 - necesară atunci când un commit este referit
- branch-uri la distanță
 - referite ca `<remote_name>/<remote_branch>`
 - `git fetch <remote_branch>` - nu realizează integrarea modificărilor în mod automat
 - `git merge <remote_name>/<remote_branch>`
 - încărcarea versiunii obținute
 - branch-uri de monitorizare
 - `git checkout -b <local_branch> <remote_name>/<remote_branch>`
 - `git checkout -track <remote_name>/<remote_branch>`, dacă `<remote_branch> = <local_branch>`
 - adăugarea unui branch pe un server la distanță
`git push <remote_name> <local_branch>[:<remote_branch>]`
 - ștergerea unui branch pe un server la distanță
`git push <remote_name> :<remote_branch>`

Crearea unei Aplicații Android în Eclipse Luna SR1a (4.4.1)



- ❑ parametri de configurare
 - Application Name – denumirea în Settings → Application Manager
 - Project Name
 - Package Name – pachetul ce conține aplicația; **identificator unic**, conservat între versiuni diferite
 - Minimum Required SDK – nivelul de API minim pe care va rula aplicația
 - Target SDK – nivelul de API maxim pentru care a fost testată aplicația
 - Compile With – nivelul de API folosit la compilare
 - Theme

- ❑ specificarea unei activități (fereastră a aplicației) + fișier xml care descrie interfața grafică

Testarea Exemplelor de Aplicații Android



- ❑ disponibile pentru fiecare platformă în parte
- ❑ ilustrează diferite funcționalități la care are acces programatorul
- ❑ parametrii de configurare
 - nivel de API \Leftrightarrow platformă \Leftrightarrow nivel API
 - producător (AOSP, GCS)