19 — `init.sh` e Definition of Done

19 — `init.sh` e Definition of Done

> Um script que verifica o baseline do projeto antes e depois de cada sessão, em dois modos. E a definição de quando uma tarefa está de fato concluída. **Um `init.sh` que não roda é pior que nenhum.** Execute-o antes de dizer que terminou.

Por que existe

Sem baseline verificado, uma sessão começa sobre terreno já quebrado e o dev gasta uma hora depurando um erro que não era dele. O `init.sh` responde a uma pergunta em segundos: *o projeto está são agora?*

Roda no início da sessão (o terreno está limpo?) e no fim (eu quebrei algo?).

FAST e FULL

A distinção é o que faz o script ser usado de fato, em vez de pulado por demorar.

Modo	Invocação	O que roda	Quando

FAST	`./init.sh`	Lint + type-check	Toda hora, várias vezes ao dia
FULL	`FULL=1 ./init.sh`	+ `format:check` + build completo	Antes de considerar pronto

FAST precisa terminar em segundos. Se passar de ~30s, algo está no modo errado.

O script

```
#!/usr/bin/env bash
# init.sh — verifica o baseline do projeto antes/depois de uma sessao.
#
# Uso:
#   ./init.sh          # rapido (padrao): lint + type-check#   FULL=1 ./init.sh   # completo: + format:check + build
#
# Pre-requisitos:#   - Node >= 24 no PATH (gerenciado via fnm — ver harness/guides/dev-environment.md)
#   - npm install ja rodado
set -euo pipefail
FULL="${FULL:-0}"
ROOT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
echo "== {{NOME_DO_PROJETO}} — init.sh (modo $([ "$FULL" = "1" ] && echo FULL || echo FAST)) =="
echo ""
echo "--- [1/4] Node ---"node_expected=$(tr -d '[:space:]' < "$ROOT_DIR/.node-version")
node_actual=$(node -v 2>/dev/null | sed 's/^v//')
if [ -z "$node_actual" ]; then
  echo "ERRO: 'node' nao encontrado no PATH. Rode 'fnm use'."
  exit 1
fi
lower=$(printf '%s\n%s\n' "$node_expected" "$node_actual" | sort -V | head -n1)if [ "$lower" != "$node_expected" ]; then  echo "ERRO: Node v$node_actual encontrado, necessario >= v$node_expected."
  exit 1
fi
```

```
echo "Node v$node_actual OK"
echo ""
echo "--- [2/4] Lint ---"
(cd "$ROOT_DIR" && npm run lint)
echo ""
echo "--- [3/4] Type-check ---"
(cd "$ROOT_DIR" && npx tsc --noEmit)
if [ "$FULL" = "1" ]; then
  echo ""
  echo "--- [4/4] Formatacao + build ---"
  (cd "$ROOT_DIR" && npm run format:check)
  (cd "$ROOT_DIR" && npm run build)
else
  echo ""
  echo "--- [4/4] Build — PULADO (rode com FULL=1 para incluir) ---"
fi
echo ""
echo "== init.sh OK =="
```

Projeto web dentro de um monorepo

Se o web for submódulo de um monorepo, o `init.sh` fica na raiz do repositório e entra na pasta do módulo:

```
WEB_DIR="$ROOT_DIR/caminho/do/modulo"
echo "--- [2/4] Lint ---"
(cd "$WEB_DIR" && npm run lint)
```

Decisões do script

Item	Razão
`set -euo pipefail`	Aborta no primeiro erro; variável indefinida vira falha, não string vazia
Checagem de Node primeiro	Erro claro em vez de falha incompreensível do npm

Item	Razão
`npx tsc --noEmit` no FAST	Barato e pega o que o checker do dev não viu
`build` só no FULL	Demora demais para rodar a toda hora
`format:check`, não `format`	Verificação não deve reescrever arquivo

Não copie comandos de outro projeto (ex.: `./mvnw compile`, `mvn test` de um projeto fullstack Java+React). Os comandos precisam ser os reais deste projeto.

Rodar o script

```
chmod +x init.sh
./init.sh
```

No Windows, execute pelo Git Bash. Se o time for todo Windows, considere um `init.ps1` equivalente — mas mantenha um só como fonte de verdade, para não divergirem.

Execute o script depois de gerá-lo. Isso não é sugestão: um `init.sh` que nunca rodou tem probabilidade alta de estar quebrado (caminho errado, script npm inexistente, `set -e` derrubando num comando que retorna não-zero legitimamente).

Startup Workflow

Antes de escrever código:

1. `pwd` — confirme o diretório.
2. Resolva a branch atual (`git rev-parse --abbrev-ref HEAD`). Se `harness/<branch>/state/` não existir, crie a partir de `harness/_examples/` (scaffolding mecânico, não preenchimento de conteúdo).
3. Leia `harness/user_preferences.md` (se existir).
4. Leia `harness/<branch>/state/feature_list.json` e `progress.md` (se existirem).
5. `git log --oneline -5`.
6. `./init.sh`.

Se o baseline já estiver quebrado, corrija antes de implementar feature nova. Misturar correção de baseline com feature nova produz um diff que ninguém consegue revisar.

Definition of Done

Uma tarefa só está concluída quando:

- [ ] O comportamento alvo foi implementado
- [ ] `./init.sh` passa — FAST no mínimo, FULL antes de considerar pronto para revisão
- [ ] **A tela foi aberta no browser e o fluxo verificado contra o backend real**
- [ ] `harness/<branch>/state/feature_list.json` atualizado com `status` e `evidence`
- [ ] `harness/<branch>/state/progress.md` atualizado

O item do browser não é opcional

Não há suíte de testes (14). `lint` e `tsc` provam que o código compila, não que funciona. A única evidência funcional que existe neste padrão é abrir a tela e verificar o fluxo. Ver o checklist de 17.

Evidência é texto, não adjetivo

```
// RUIM — nao e' evidencia
"evidence": "Implementado e testado."
// BOM — verificavel"evidence": "Implementado em 2026-08-18. ./init.sh FULL verde. Tela cadastro/fornecedor aberta em dev:busca com filtro 'ACME' retornou 3 registros; salvar com CNPJ invalido bloqueou no zod; salvar validomostrou notificacao verde e o registro apareceu na tabela apos rebusca. Sem erro no console."
```

Não marque como `passing` só porque o código foi escrito. É o erro mais comum, e o que torna o `feature_list.json` inútil.

Fim de sessão

1. Atualizar `harness/<branch>/state/feature_list.json` (status + evidência): se algo chegou a `passing`, mover a entrada de `features` para `completed` **no mesmo arquivo** — sem cópia

para nenhum arquivo global, não existe mais (ver 18).
2. Atualizar `harness/<branch>/state/progress.md`.
3. Atualizar `harness/<branch>/state/session-handoff.md` com o resultado da verificação.
4. Commit descritivo com o repositório em estado seguro — inclui `state/`, `plans/`, `specs/` e `handoffs/` da branch, tudo versionado.
5. **A próxima sessão deve conseguir rodar `./init.sh` imediatamente.**

O passo 5 é o teste do fim de sessão: se a próxima pessoa não consegue verificar o baseline sem antes consertar algo, a sessão não fechou.

Se o projeto adotar testes

Se 14 for revisto e o projeto passar a ter suíte:

1. Adicione `npm test` ao bloco FULL do `init.sh`.
2. Adicione "testes passando" à Definition of Done.
3. Mantenha o item da verificação no browser — teste unitário não prova integração com o Curio.

Revision #2
Created Thu, Aug 20, 2026 5:10 PM by Geraldo Barbosa
Updated Tue, Aug 25, 2026 4:56 PM by Geraldo Barbosa