LABORATÓRIO DE ENGENHARIA DE SOFTWARE

Git em equipe, da contribuição à entrega

Quatro exercícios avançados e não resolvidos sobre Git, GitHub, GitHub Pages, DNS e Cloudflare.

Uma sequência de investigação técnica

Cada desafio começa com evidências incompletas e termina em uma decisão verificável.

01

Estado e ponteiros

02

Revisão e integração

03

Pages, DNS e HTTPS

04

Incidente ponta a ponta

Caderno de atividades | Sem soluções | Agosto de 2026

Como usar este caderno

Os quatro exercícios formam uma única progressão. Começamos dentro do repositório local, atravessamos o *fork* e o *pull request*, publicamos a *main* no GitHub Pages e terminamos investigando um incidente que alcança DNS, HTTPS, proxy e cache. A estudante não receberá uma sequência pronta de comandos: deverá reconstruí-la a partir do estado observado.

> **Contrato do material**
>
> Este PDF contém somente enunciados. Não há solução, gabarito, roteiro oculto nem resultado esperado além dos critérios de aceitação declarados. Toda resposta deverá justificar decisões, prever efeitos e apresentar evidências que outra pessoa consiga conferir.

Base conceitual

Artigo	Conhecimento mobilizado
Do Git ao domínio próprio, publicando um site estático com GitHub Pages e Cloudflare	Repositório, *commit*, áreas de trabalho, colaboração, GitHub Pages, domínio, DNS, Registro.br, HTTPS, Cloudflare, DNSSEC, proxy e cache.
Do branch ao pull request, colaboração distribuída com Git e GitHub	*Branch*, HEAD, *fork*, *clone*, *origin*, *upstream*, revisão, proteção da *main*, conflitos e estratégias de integração.

Regras de resposta

- Registre o estado inicial antes de propor qualquer mudança. Um comando sem interpretação não constitui evidência.
- Diferencie o repositório oficial, o *fork*, o repositório local e as referências de acompanhamento. Use nomes completos sempre que houver ambiguidade.
- Preserve todo trabalho não registrado. São proibidos atalhos destrutivos, inclusive `git reset --hard`, substituição cega de arquivos e *push* forçado.
- Valide uma camada por vez. Uma interface administrativa informa intenção; a linha de comando e a resposta pública comprovam o estado observado.
- Entregue comandos, saídas relevantes, capturas ou URLs e uma explicação curta do que cada evidência demonstra.

Critérios comuns de avaliação

Critério	Peso	O que será observado
Modelo do estado	25%	Objetos, ponteiros, repositórios e camadas foram distinguidos sem ambiguidade.
Correção técnica	25%	A sequência proposta alcança o objetivo sem introduzir uma falha nova.
Segurança operacional	25%	O plano preserva história, trabalho local, disponibilidade e possibilidade de reversão.
Evidência	25%	Outra pessoa consegue reproduzir a verificação e relacionar cada saída a uma conclusão.

Os nomes e endereços pertencem ao cenário acadêmico. O IPv4 `192.0.2.10` é reservado para documentação. Nenhuma atividade exige alterar um domínio real.

01 Autópsia de um fork divergente

Estado local | staging area | HEAD | origin | upstream | sincronização

Ana trabalha no *fork* de um site acadêmico. Ela abriu uma ramificação de acessibilidade, interrompeu a tarefa para corrigir o *README* e, enquanto isso, duas contribuições chegaram à `main` oficial. Antes de tocar no histórico, precisamos descobrir onde cada mudança vive.

Dados do cenário

Componente	Estado observado
Repositório oficial	`professor/equipe-site` no GitHub
Fork de Ana	`ana/equipe-site` no GitHub
Repositório local	Clone de `ana/equipe-site`; nenhuma operação destrutiva foi executada
Tarefa	Apenas a acessibilidade do menu em `index.html` pertence ao escopo
Trabalho paralelo	A edição de `README.md` pertence a outra tarefa; `notas.txt` não deve ser versionado
Regra da main	A base local deve espelhar a *main* oficial e avançar somente em linha reta

Saída dos remotos e da árvore de trabalho

```
PS> git remote -v
origin    github.com/ana/equipe-site.git       (fetch)
origin    github.com/ana/equipe-site.git       (push)
upstream  github.com/professor/equipe-site.git (fetch)
upstream  github.com/professor/equipe-site.git (push)

PS> git status --short --branch
## feature/acessibilidade...origin/feature/acessibilidade [ahead 2]
 M README.md
M  index.html
?? notas.txt
```

Na forma curta de `git status`, a primeira coluna descreve a *staging area* e a segunda descreve a árvore de trabalho. O grafo abaixo foi obtido logo depois de `git fetch upstream`.

```
* f2 (HEAD -> feature/acessibilidade) Ajusta o foco do menu
* f1                                  Adiciona descrição aos links
| * d3 (upstream/main)                Integra a página de contato
| * c2                                Corrige caminhos de imagens
|/
* b1 (main, origin/main, origin/feature/acessibilidade)
|                                     Publica a página inicial
* a0                                  Configura o repositório
```

Restrições

Não apague nem inclua em um *commit* da tarefa as mudanças de `README.md` e `notas.txt`. Não reescreva a história já publicada, não use *push* forçado e não crie outro *fork*. Ao final, a ramificação de acessibilidade deverá ser publicada no *fork* e comparável com a *main* oficial atual.

Problema

1 Construa um inventário de *HEAD*, `main`, `origin/main`, `upstream/main`, da ramificação local e de sua referência remota. Para cada item, informe onde vive, para qual *commit* aponta, qual operação pode movê-lo e se representa informação atual ou apenas a última observação conhecida.

2 Decodifique as três linhas de arquivos do `git status`. Separe o que já está preparado, o que ainda está apenas na árvore de trabalho e o que não está rastreado. Explique por que trocar de ramificação ou iniciar uma mesclagem neste estado merece uma decisão explícita.

3 Escreva uma sequência conservadora de operações que preserve todo o trabalho, sincronize a `main` local com `upstream/main` por avanço em linha reta, atualize `origin/main`, traga a base oficial recente para `feature/acessibilidade` e publique a tarefa. Inclua pontos de inspeção antes e depois de cada mudança de estado.

4 Desenhe o grafo esperado depois da sincronização. Identifique os quatro campos do *pull request*, isto é, o repositório e a ramificação de *base*, além do repositório e da ramificação de *head*. Justifique a direção escolhida.

5 Defina um teste de escopo que prove que o *pull request* contém apenas a alteração de acessibilidade. A resposta deverá distinguir o *diff* da árvore de trabalho, o *diff* preparado e o *diff* da proposta em relação ao ponto comum de divergência.

Evidências obrigatórias

- Um mapa antes e depois, com os ponteiros e os hashes abreviados.
- As saídas interpretadas de `git status`, `git remote -v` e `git log --oneline --decorate --graph --all`.
- O comando usado para comparar a contribuição com a base comum e uma explicação do escopo encontrado.
- Uma tabela de risco com, no mínimo, perda de trabalho local, *commit* na ramificação errada, referência remota desatualizada e *push* para o destino incorreto.

02

Um pull request que ainda não pode entrar

base e head | code review | checks | conflito | ruleset | estratégia de integração

Carla abriu o *pull request* 42 para acrescentar uma página de contato. A proposta parece pequena na prévia, mas a *main* avançou, uma verificação falhou e a política do repositório descarta aprovações antigas quando novos *commits* chegam. A estudante deverá decidir o que bloqueia a integração e como amadurecer a mesma proposta.

Identidade e política do pull request

Campo	Valor
Base	`professor/equipe-site:main`
Head	`carla/equipe-site:feature/formulario-contato`
Escopo declarado	`contato.html` e um link no menu de `index.html`
Estratégia permitida	Somente *squash and merge*
Ruleset	Exige dois *approvals*, descarta aprovações obsoletas, exige conversas resolvidas, exige o *check* `links`, bloqueia exclusão e *force push*

Grafo e estado de revisão

```
* p3 (HEAD -> feature/formulario-contato) Corrige o texto do formulário
* p2                                     Adiciona validação dos campos
* p1                                     Cria a página de contato
| * m2 (upstream/main)                   Adiciona recursos de acessibilidade
| * m1                                   Reorganiza o menu principal
|/
* b0                                     Publica a versão 1.0

Reviews: 1 Approve anterior a p3 | 1 Request changes | 1 conversa aberta
Checks:  html PASS | links FAIL
Mergeability: conflito em index.html
```

Trecho do check obrigatório

```
links / verify-links
ERROR index.html:47
href="/contatoo.html" -> HTTP 404
1 broken link, 0 warnings
```

Região em conflito depois de trazer upstream/main

```
<<<<<<< HEAD
<li><a href="/contato.html">Contato</a></li>
=======
<li><a href="/acessibilidade.html">Acessibilidade</a></li>
>>>>>>> upstream/main
```

Problema

1. Audite a possibilidade de integração. Relacione cada regra do *ruleset* a uma evidência atual e classifique o item como satisfeito, bloqueado ou indeterminado. Não trate todos os impedimentos como um único conflito.
2. Planeje a atualização da mesma ramificação de *head* com a `main` oficial. Sua resposta deverá incluir a leitura das três versões do conflito, o critério usado para produzir o conteúdo final, a inspeção de marcadores remanescentes, o novo *commit* e a publicação sem abrir outro *pull request*.
3. Descreva uma revisão reproduzível do *diff*. Verifique escopo, navegação, semântica HTML, comportamento do formulário, ausência de remoções acidentais e correspondência entre o resultado local e os *checks* do GitHub.

4 Explique o que acontecerá com a aprovação anterior depois de novos *commits*, quais conversas precisam de ação e quais condições devem estar verdadeiras antes que o botão de integração seja usado.

5 Compare, por meio de pequenos grafos, como *merge commit*, *squash and merge* e *rebase and merge* representariam os três *commits* da proposta. Em seguida, prepare o título e o corpo do único *commit* compatível com a política do cenário.

6 Produza uma decisão final de revisão em até 180 palavras. A decisão deverá citar evidências, separar defeitos de bloqueio técnico de observações não bloqueantes e declarar se a proposta está ou não pronta naquele instante.

Evidências obrigatórias

- Matriz regra, evidência, estado e ação necessária.
- Grafo antes da atualização, grafo da ramificação atualizada e grafo esperado na `main` depois do *squash*.
- Saídas interpretadas do *diff* de três pontos, da inspeção de conflitos e dos *checks* locais.
- Texto da revisão e mensagem final de integração, sem executar a integração enquanto existir um bloqueio.

03 O site existe, mas o domínio não prova isso

GitHub Pages | fonte de publicação | DNS | domínio personalizado | HTTPS | proxy

A equipe afirma que a propagação ainda não terminou. Contudo, o site de usuário, o domínio personalizado e o ápice falham de maneiras diferentes. O cenário contém defeitos independentes, e esperar não altera nenhum valor publicado incorretamente.

Estado do repositório e do GitHub Pages

Item	Estado observado
Repositório	`equipe/equipe.github.io`, ramificação `main`
Arquivos	`/index.html`, `/CNAME` e `/assets/site.css`
Conteúdo de CNAME	`www.projetoexemplo.com.br`
Fonte do Pages	Deploy from a branch: `main` e pasta `/docs`
Implantação 153	Concluída para o *commit* `9f4c2a1`
Custom domain	`www.projetoexemplo.com.br` configurado; *Enforce HTTPS* indisponível

Zona ativa no Cloudflare, ainda em DNS only

Tipo	Nome	Valor	Proxy
A	@	185.199.108.153	DNS only
A	@	185.199.109.153	DNS only
A	@	192.0.2.10	DNS only
CNAME	www	equipe.github.io	DNS only

Para este cenário, a especificação fornecida pela disciplina exige quatro registros `A` no zone apex: `185.199.108.153`, `185.199.109.153`, `185.199.110.153` e `185.199.111.153`. O valor `192.0.2.10` é apenas um endereço de documentação.

Observações coletadas por duas redes

```
PS> Resolve-DnsName projetoexemplo.com.br -Type NS
amy.ns.cloudflare.com    art.ns.cloudflare.com

PS> Resolve-DnsName www.projetoexemplo.com.br -Type CNAME
www.projetoexemplo.com.br -> equipe.github.io

PS> Resolve-DnsName projetoexemplo.com.br -Type A
185.199.108.153  185.199.109.153  192.0.2.10

PS> curl.exe -I https://equipe.github.io/
HTTP/2 404

PS> curl.exe -I https://www.projetoexemplo.com.br/
HTTP/2 404

Rede A, ápice: timeout de conexão
Rede B, ápice: HTTP/2 301 -> https://www.projetoexemplo.com.br/
```

Janela de mudança

A equipe pode alterar a configuração do GitHub Pages, o conteúdo da *main* e os registros DNS da zona. Toda mudança no estado do proxy deverá ser justificada por uma evidência anterior e seguida por um novo teste. HSTS e regras amplas de cache estão fora desta janela.

Problema

1. Construa uma árvore de falhas com quatro fronteiras: repositório e fonte de publicação, GitHub Pages, DNS autoritativo e HTTPS do domínio personalizado. Relacione cada observação à fronteira que ela realmente testa.
2. Identifique todos os defeitos independentes presentes nos dados. Para cada um, explique por que esperar pela propagação pode ou não alterar o resultado e qual teste separa erro de configuração de cache ainda não expirado.
3. Determine e justifique uma ordem de reparo com pontos de parada. Compare manter os registros em DNS only com ativar o proxy em cada etapa e explique qual evidência autoriza a mudança seguinte sem esconder uma falha da origem.
4. Monte uma matriz de validação contendo comando, camada testada, resposta mínima aceitável, resposta que ainda indica falha e decisão seguinte. Inclua consultas de `NS`, `A` e `CNAME`, cabeçalhos HTTP e uma verificação do marcador textual publicado no *commit* `9f4c2a1`.
5. Defina o comportamento canônico entre o ápice e `www`. A resposta deverá escolher uma única autoridade para o redirecionamento, limitar o número de saltos e evitar que duas plataformas imponham sentidos opostos.
6. Especifique o pacote de aceitação que será entregue ao professor. Uma captura do site sem consulta DNS, cabeçalho HTTP e identificação do *commit* não será suficiente.

Evidências obrigatórias

- Árvore de falhas e lista de defeitos independentes, cada qual ligado a uma observação.
- Plano ordenado com condição de avanço e possibilidade de reversão em cada camada.
- Matriz de comandos e critérios de aceitação, executada antes e depois das mudanças.
- Prova de que a versão publicada corresponde à `main` aceita, não apenas a uma resposta antiga do navegador ou do cache.

04 Incidente na cadeia completa de entrega

pull request | Pages | DNSSEC | Cloudflare | TLS | redirecionamento | cache | rollback

Uma equipe integra a nova versão do site e, na mesma tarde, transfere a autoridade DNS para o Cloudflare. O conteúdo correto chega à origem, mas parte das visitantes recebe `SERVFAIL`, outra parte entra em um laço de redirecionamento e uma terceira ainda vê HTML antigo. A tentativa de correção direta na *main* é recusada.

Linha do tempo observada

Hora	Evento
13:40	O *pull request* 27 é integrado por *squash*; a `main` oficial avança para `s7a91e2`.
13:43	A implantação do GitHub Pages termina com sucesso; `https://equipe.github.io/` mostra o marcador `release-2026-08`.
13:50	A zona é adicionada ao Cloudflare. Os registros web são importados, ainda em DNS only.
14:05	Os servidores autoritativos são trocados no Registro.br sem remover antes o registro DS do provedor DNS anterior.
14:18	Um resolver com validação DNSSEC responde `SERVFAIL`; um teste sem validação encontra os novos servidores do Cloudflare.
14:25	O proxy é ativado antes da validação do domínio; o modo TLS escolhido é Flexible.
14:32	O GitHub redireciona o ápice para `www`; uma regra do Cloudflare redireciona `www` para o ápice.
14:45	Uma regra `Cache Everything` para `/*` fixa HTML por 24 horas.
15:10	Uma estudante tenta corrigir a página por *push* direto e depois por `git push --force`; o *ruleset* recusa ambas as operações.

Amostra de evidências

```
PS> curl.exe -s https://equipe.github.io/ | findstr release
<meta name="release" content="release-2026-08">

PS> Resolve-DnsName www.projetoexemplo.com.br -Type A
Resolve-DnsName: DNS name does not exist or validation failed (SERVFAIL)

PS> curl.exe -I -L --max-redirs 5 https://projetoexemplo.com.br/
curl: (47) Maximum (5) redirects followed

PS> curl.exe -I https://www.projetoexemplo.com.br/
HTTP/2 200
cf-cache-status: HIT
age: 43120

PS> curl.exe -s https://www.projetoexemplo.com.br/ | findstr release
<meta name="release" content="release-2026-07">
```

Objetivo operacional

Restabelecer disponibilidade e uma única versão observável sem reescrever a história oficial. A equipe deverá separar a cadeia de desenvolvimento da cadeia de entrega, reduzir o número de variáveis em cada teste e manter um caminho de reversão documentado.

Problema

1. Desenhe duas cadeias causais. A primeira deverá começar na ramificação de tarefa e terminar na origem do GitHub Pages. A segunda deverá começar na query DNS da visitante e terminar na resposta entregue. Posicione cada evento da linha do tempo na cadeia correspondente.
2. Classifique cada sintoma como causa primária, consequência, agravante ou tentativa de correção recusada. Relacione o sintoma à camada responsável e cite a evidência que sustenta a classificação. Um mesmo incidente pode possuir mais de uma causa primária.

3 Escreva um *runbook* de restauração com ordem, responsável, comando ou painel, teste de saída, condição de avanço e *rollback*. O plano deverá tratar explicitamente a confiança DNSSEC antiga, a delegação, a linha de base em DNS only, o certificado da origem, o modo TLS, o laço de redirecionamento e o cache de HTML.

4 Defina como uma correção de conteúdo urgente atravessará a proteção da `main`. Compare um novo *pull request* corretivo com a reversão de um *commit* já integrado e explique quando cada mecanismo preserva melhor a rastreabilidade. Não use *push* forçado.

5 Construa uma matriz de aceitação ponta a ponta. Ela deverá comprovar a `main` protegida, a implantação do *commit* escolhido, a cadeia `DS/DNSKEY`, os servidores autoritativos, o HTTPS válido entre visitante, proxy e origem, um único redirecionamento canônico, a passagem pelo Cloudflare e a ausência de HTML obsoleto.

6 Redija um pós-incidente entre 250 e 350 palavras. Separe impacto, cronologia, causas técnicas, fatores de processo, ações imediatas e ações preventivas. Não atribua a falha a uma pessoa; relacione cada ação preventiva a um mecanismo verificável.

7 Proponha um protocolo permanente de mudança com, no mínimo, proteção da *main*, revisão, validação do Pages, janela de DNS, migração de DNSSEC, ativação gradual do proxy, política de redirecionamento e escopo de cache. Indique qual evidência autoriza a passagem entre etapas.

Evidências obrigatórias

- Diagrama das duas cadeias com dependências, sinais observáveis e fronteiras de responsabilidade.
- Tabela cronológica de causas e sintomas, sem confundir correlação temporal com causalidade.
- Runbook executável com decisões de avanço e reversão.
- Matriz final de aceitação e pós-incidente, ambos ligados aos mesmos critérios técnicos.

Matriz de cobertura

A tabela permite conferir se a entrega percorreu os dois artigos como um sistema único. Uma marca indica que o tópico constitui parte explícita do problema, não que uma frase isolada sobre ele seja suficiente.

Tópico	Ex. 1	Ex. 2	Ex. 3	Ex. 4
Árvore de trabalho, staging area e commit	Principal	Apoio	Apoio	Apoio
Branch, HEAD e grafo	Principal	Principal		Apoio
Fork, clone, origin e upstream	Principal	Principal		Apoio
Pull request, revisão e ruleset	Apoio	Principal		Principal
Conflito e estratégia de integração	Principal	Principal		Principal
GitHub Pages e fonte de publicação		Apoio	Principal	Principal
Registro.br, DNS e domínio personalizado			Principal	Principal
Cloudflare, TLS, proxy e cache			Principal	Principal
DNSSEC, incidente e rollback			Apoio	Principal

Checklist de submissão

- Cada exercício contém um modelo explícito do estado inicial e do estado final pretendido.
- Os comandos aparecem em ordem e são acompanhados pela interpretação da mudança de estado que produzem.
- Toda ação potencialmente destrutiva foi removida ou substituída por uma alternativa com preservação e reversão.
- As evidências distinguem interface, repositório, implantação, DNS, HTTPS, proxy e cache.
- Nenhuma conclusão depende exclusivamente de uma captura de tela, do cache do navegador ou da frase funcionou na minha máquina.
- Os quatro exercícios foram respondidos sem consultar uma solução fornecida por este caderno, porque ela não existe aqui.

Encerramento

O objetivo não será decorar uma procissão de comandos. Será demonstrar que cada mudança possui um lugar, cada integração possui uma regra e cada falha deixa sinais em uma camada específica.

Material complementar aos artigos publicados em frankalcantara.com. Preparado para uso acadêmico em Engenharia de Software.